THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS

LIGHTWAVEPRO

an Avid Media Group, Inc. newsletter

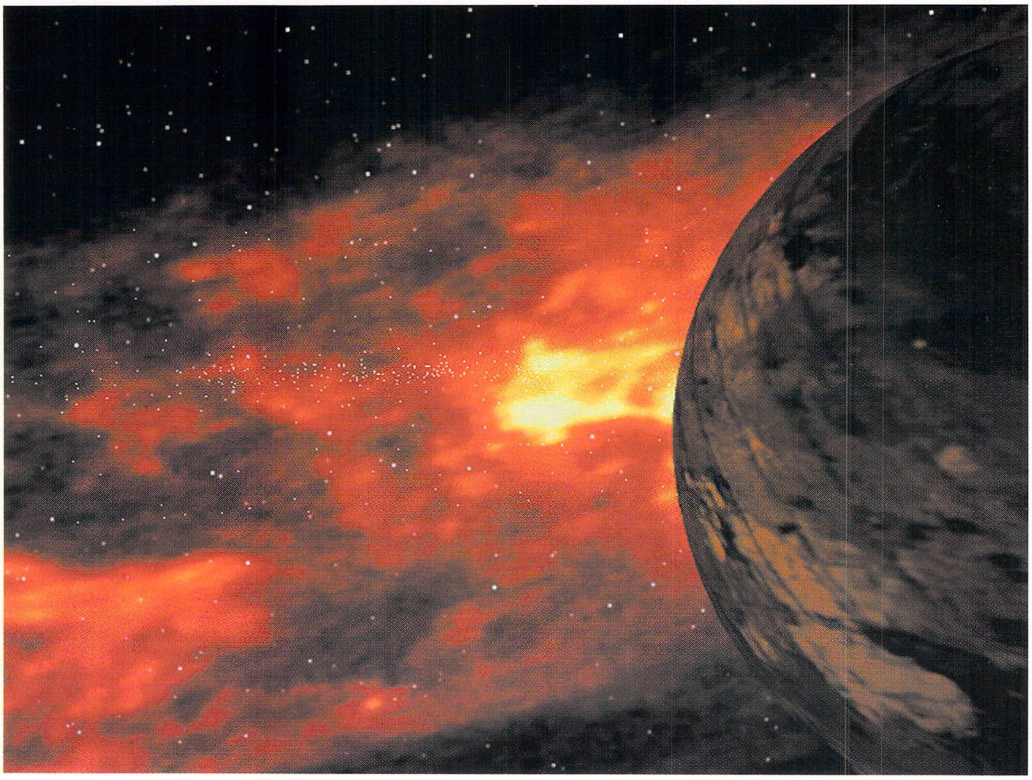
Volume 3 Issue Number 2 • February 1995 • $8.00

Destroy Your Own Spaceship

Inside:

- Land Vehicle Movement
- Rendering Algorithms—Part II
- You Bonehead! A Beginner's Look at Bones

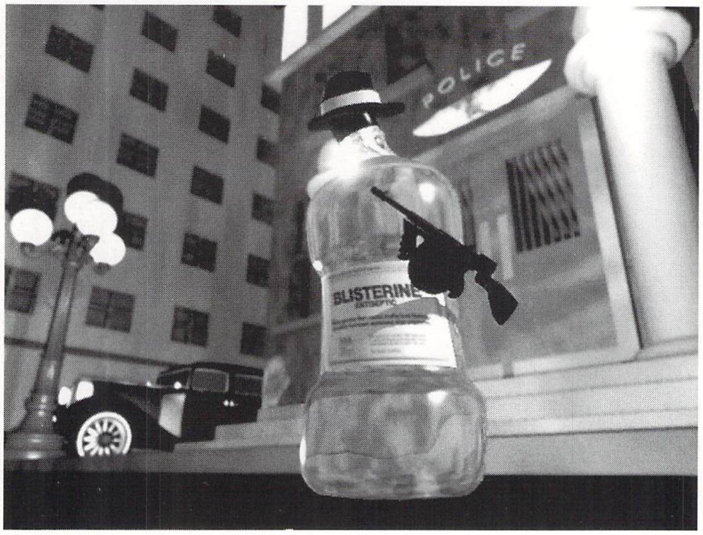
What's on the LIGHTWAVEPRO Disk? See Page 13



Bad Space

The galactic equivalent of a scary neighborhood. The dark planet with a deep orange key light further suggests a sinister locale. See "Dead in Space," page 6.

Copyright 1994,1995 PTEN Consortium, Inc.



Blisterine

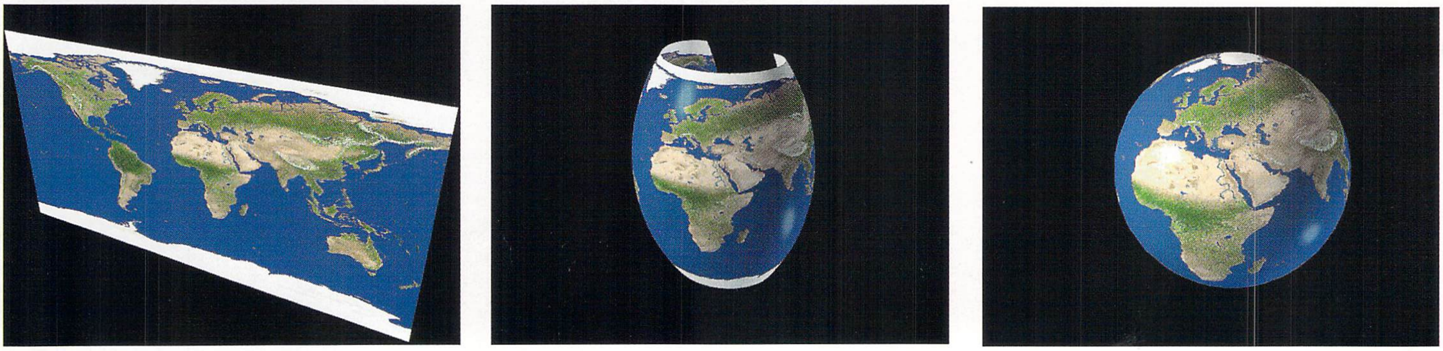
A still from the Blisterine animation. Bones are used to create effective character movements. See "You Bonehead," page 8.

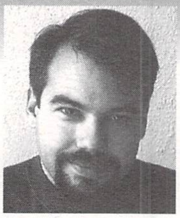
Copyright 1994 Dan Ablan

World Wrap

The progression of morphing a flat plane to a sphere using a cylinder as an intermediate shape. See "Mighty Morphin' Morphing Tricks," page 12.

Copyright 1995 James G. Jones.





EDITOR'S MESSAGE

by John Gross

You're probably sick of hearing about the Internet. After all, it's one of the trendiest computer things around. I've seen magazines, books, software and television shows devoted entirely to this service. I would say about the only thing I have heard or seen more about is O.J. Simpson (and believe me— living in Los Angeles, that's getting pretty tiresome).

But for as much hoopla as I've experienced about the net, I've also experienced the net itself, and I tell you, there are some great resources of information. You can find just about anything on the net if you know where to look—or even if you don't know where to look, but aren't afraid of trying.

There are newsgroups on the net that cover just about any topic you could dream of, and some you couldn't. And the nice thing is, the net is not censored, like commercial on-line services.

What does this have to do with LightWave animators? A lot. I know I've mentioned it before, but I get so many people asking questions about the LightWave information on the net that it's worth repeating.

There are a couple of great ways of accessing LightWave info on the net. First of all, there is a LightWave mailing list. This is a group of people that subscribe to this "list" and everyone gets the mail that is sent to it. You can participate by asking or answering questions, or by just following along. Topics range from the simple to the complex, and most have to do with LightWave and Modeler, but every once in a while, a stray one gets thrown in. To subscribe to the list, simply send an e-mail message to listserv@netcom.com. In the body of the message state "subscribe lightwave-l your address" (no quotes). In a short time, you will start receiving mail (be prepared!).

Because of the popularity of the LightWave mailing list, a LightWave newsgroup was started. This allows you to see topics, most having to do with LightWave, that you can either choose to read or ignore. The newsgroup is called comp.graphics.packages.lightwave. Both the newsgroup and the mailing list are wonderful sources of information, tips, tricks, problem-solvers and technical answers, and are frequented by a large amount of very talented people. You can pretty much be assured of an answer to any type of question you may have.

see Editor's Message, page 7

TABLE OF CONTENTS

4 Land Vehicle Movement

by Joe Dox

Your objects are spinning out of control?! Learn how to give the wheels on your moving land vehicles realistic rotating motion.

6 Dead in Space

by Mojo

Embark on a step-by-step journey through one of *Babylon 5's* most explosive scenes.

8 You Bonehead!

by Dan Ablan

Don't be fooled by the seeming complexity of Bones—these free-form deformation tools can bring life to your inanimate objects. Animate an object with the fluid characteristics of the popular Listerine bottle.

10 Popular 3D Algorithms

by William Frawley

In this follow-up to last month's look at Z-buffer rendering, explore the subtleties of raytracing and radiosity techniques.

12 Mighty Morphin' Morphing Tricks

by James G. Jones

It's morphing time—3D object morphing, that is. Discover some of the many functions of this useful process.

14 Reader Speak

by John Gross

An explanation of modifying configuration files for LightWave and Modeler.

LIGHTWAVEPRO

an Avid Media Group, Inc. newsletter

Editor	John Gross
Managing Editor	Jim Plant
Editorial Coordinator	Joan Burke
Associate Editor	Corey Cohen
Art Director	Helga Nahapetian Taylor
Art/Production Coordinator	Kristin Fladager
Production	Sergio "Berimbau" Miller
Circulation Director	Sherry Thomas-Zon
Circulation Assistants	Tracy Ann-Sparks
	Debra Goldsworthy
Contributing Writers	Dan Ablan
	Joe Dox
	William Frawley
	James G. Jones
	Mojo
Group Publisher	Michael D. Kornet

Editorial Offices: Avid Media Group, Inc.
273 N. Mathilda Avenue
Sunnyvale, CA 94086
Telephone (408) 774-6770
Fax (408) 774-6783
John Gross can be reached electronically at:
jgross@netcom.com (Internet)
71740,2357 (CompuServe)
Printed in the USA© 1995 Avid Media Group, Inc.

LIGHTWAVEPRO (Vol. 3, No. 2); (ISSN 1076-7819) is published monthly by Avid Media Group, Inc., 273 N. Mathilda Ave., Sunnyvale, CA 94086-4830. A one-year subscription (12 issues) in the U.S. and its possessions is $48 (U.S.); Canada/Mexico, $60 (U.S.); overseas, $84 (U.S.). To subscribe, call toll-free 1-800-322-2843. Allow 4 to 6 weeks for first issue to arrive. Second-class postage rate paid at Sunnyvale, CA and additional mailing offices. POSTMASTER: Send address changes to *LIGHTWAVEPRO*, 273 N. Mathilda Ave., Sunnyvale, CA 94086-4830.

Are you interested in writing for *LIGHTWAVEPRO* or submitting images? If so, contact us at our offices or electronically.

Avid Media Group, Inc., its employees or freelancers are not responsible for any injury or property damage resulting from the application of any information in *LIGHTWAVEPRO*.

About the cover: A Narn cruiser gets sliced and diced by the Shadowmen in the *Babylon 5* episode "Revelations." The scene was designed and created by Mojo for Foundation Imaging. The Narn cruiser was built by Paul Beigle-Bryant. All images © 1994, 95 PTEN Consortium, Inc.

Land Vehicle Movement

Spinning Your Wheels

by Joe Dox

One of the biggest challenges with animating land vehicles is wheel movement. This includes animating a car driving down a road at 10 mph and an aspirin tablet rolling across a 3D chart. To put it simply, the rotation of a particular object must be directly related to its velocity. Unlike a spaceship or a submarine, the wheels of a moving vehicle should have realistic rotating motion. Other issues with animating land vehicles involve body movement and track marks left behind by the rotating wheels.

The vehicle shown in Figure 1 is a military-style Hummer, otherwise known as a HUM-V. My goal was to create realistic movement of the wheels and body while it cruised across the desert. First, I'll describe how the Hummer's movement was set up. I will then discuss how to acquire rotational values for wheels moving at a particular speed, and how to create tracks left behind by the moving tires. Sorry, I won't be explaining how I modeled the Hummer—maybe in my next tutorial.

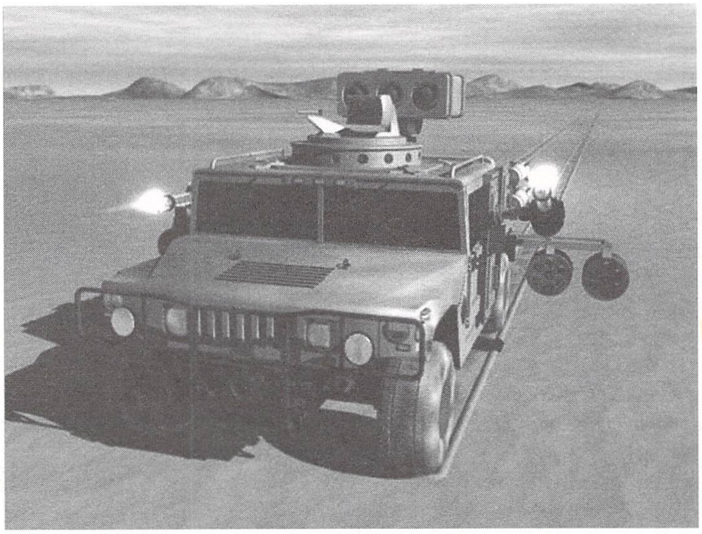
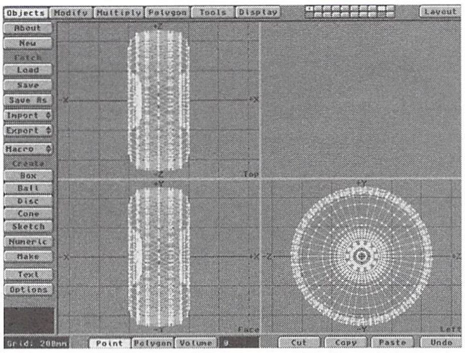


Figure 1: "Full Metal Hummer" uses simple geometric formulas to create realistic velocity and wheel rotation.

Setting Up a Vehicle Scene

When creating a scene with a moving vehicle, the objects must consist of at least the vehicle body and four wheels. One might say that the wheels should be parented to the car body. Well, they could be, but I recommend the following process:

- Load the vehicle body and the four wheels. The left and right wheels should be saved as individual objects from Modeler. Each wheel should have its pivot point in its exact center, so you'll have a smooth, even rotation. To do this, manually move the pivot point of each wheel from inside LightWave. I recommend loading (or importing) a wheel into Modeler, then selecting the **Macro** button (**Objects** menu) and executing the Center macro. The object will be placed in the exact center of the Modeler environment (Figure 2). Save (or export) the wheel and repeat this process for the others.
- Back in Layout, the next step involves creating a null object by hitting the **Add Null Object** button (**Objects** panel). Parent the car body and each wheel to this null object. After every object has been parented to the null object, you will need to position each wheel in relation to the car body. In other words, put the wheels where they belong (Figure 3). Don't forget to create a keyframe for frame 0 after each wheel has been put into position. The vehicle is now ready to be animated.



Objects
Modify
Multiply
Polygon
Tools
Display
Layout
About
New
Fetch
Load
Save
Save As
Import
Export
Macro
Create
Box
Ball
Disc
Cone
Sketch
Numeric
Make
Text
Options
Top
Face
Left
Point
Polygon
Volume
Cut
Copy
Paste
Undo

Grid: 200mm

Figure 2: Wheels are centered in Modeler to make certain there are no "bumps" in their rotation.

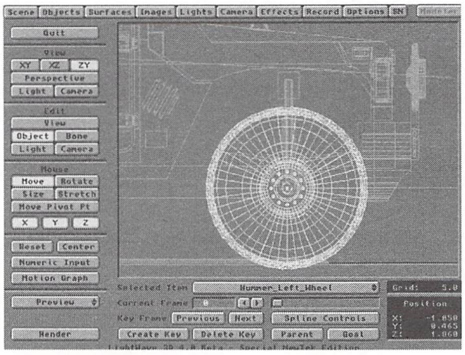
The Hummer scene shown in Figure 1 was set up in this manner. I simply loaded the Hummer body and the four individual wheels. After parenting them all to a null object, I positioned each wheel under a wheel well (Figure 3). The advantage of parenting each object to a null object is that you can move the objects independently of each other. Effects like front-wheel braking lockup and body roll can be achieved by simply selecting the object and manipulating it in the desired manner.

Moving the Vehicle

OK, we have a vehicle. Let's say we want to move it at 30 mph for five seconds. There are two things we need to compute:

1. The distance the vehicle travels in 5 seconds at 30 mph.
2. The rotational value per second (degrees/30 frames) of each wheel at 30 mph.

First, we must establish the fact that, in this article, I will be measuring distance over time as miles per hour, not kilometers per hour. But fear not, road-warriors: The formulas that I will be using are completely interchangeable between miles per hour and kilometers per hour.



Scene
Objects
Surfaces
Images
Lights
Camera
Effects
Record
Options
SN
Quit
View
XY
XZ
ZY
Perspective
Light
Camera
Edit
View
Object
Bone
Light
Camera
Mouse
Move
Rotate
Size
Stretch
Move Pivot Pt
X
Y
Z
Reset
Center
Numeric Input
Motion Graph
Preview
Render
Key Frame
Previous
Next
Spline Controls
Create Key
Delete Key
Parent
Goal

Selected Item: Hummer_Left_Wheel
Current Frame: 0
Grid: 5.0
Position
X: -1.050
Y: 0.465
Z: 1.060

Figure 3: A wheel is positioned in one of Hummer's wheel wells.

Computing the Vehicle's Velocity

Let's begin. To get the value of No. 1 (above), we must compute meters/second (M) with a given value of 30 mph. The formula is as follows:

$M = (((mph/60)\times 5280)/60)\times 12)/39.37$

For those of you who would rather work in the metric system, the formula is:

$M = (((kph/60)\times 1000)/60$

To better understand this formula, let's chop it up. First, mph/60 x 5,280 will give you feet/minute, given that 5,280 feet = 1 mile. Next, divide that quantity by 60. This will give you feet/second. Multiplying this value by 12 provides inches/second. If 39.37 inches = 1 meter, divide inches/second by 39.37. This total is a value for meters/second, or in LightWave terms, meters/30 frames.

In our example, the vehicle is traveling at 30 mph. So our formula would look like:

$M = ((((30/60)\times 5280)/60)\times 12)/39.37$
$M = [(2{,}640/60)\times 12]/39.37$
$M = (44\times 12)/39.37$
$M = 528/39.37$
$M = 13.4112$ meters/30 frames.

After you do the math for 30 mph, the vehicle will travel 13.411 meters in 30 frames. The next step is simple:

- Place the vehicle at its starting position and create keyframe 1. Move the vehicle (along the Z axis for this example) to a distance five times greater than the value we computed (5 seconds): 13.411 x 5 = 67.05 meters. Create keyframe 150. That's it! The vehicle is moving at 30 mph. Oh, don't set any spline controls. Acceleration and deceleration is something I'll discuss another time.

Computing Wheel Rotation

Now that we have the body moving at a "perfect" 30 mph, we need to rotate the wheels to match that speed. Guessing the rotational value of the wheels and rendering previews is something you could get away with, but slower speeds would be more difficult to match manually. I know—I've tried it. A quick glance at Figure 1 shows that the Hummer has huge tires with a well-defined tread pattern. It would be very easy for anyone to figure out that the wheels do not match the vehicle's speed.

We first need to compute the circumference of the wheel. The circumference of a circle is determined by a well-known formula:

$C = 2\times\pi\times radius$

- To get the radius of your particular wheel, go into Modeler, import the wheel, select the **Measure** button (**Display** menu), then measure the distance from the center of the wheel to the edge. In the case of the Hummer scene, the radius of a wheel is 0.4712 meters. (I know, that's a pretty big wheel, but a Hummer is no Suzuki SideKick.) With a radius of 0.4712 meters, the circumference of the wheel is:

$C = (2\times 3.14159)\times 0.4712 = 2.9606$

Remember, our goal is to acquire degrees of rotation for every 30 frames. With M = meters/seconds, and C = circumference of a wheel, the formula to get degrees of rotation/second (D) is:

$D = (M/C)\times 360$

Thus,

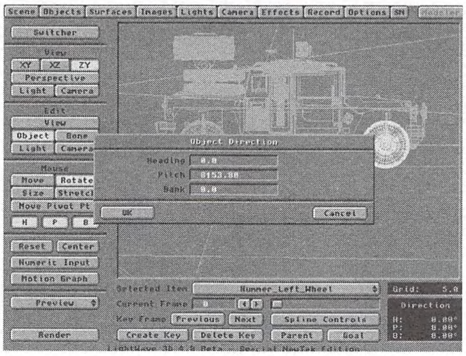
$D = (13.4112/2.9606)\times 360$
$D = (4.5298\times 360)$
$D = 1630.76$

Now that we have D, which equals the number of degrees the wheel rotates in one second, we must multiply that value by the number of seconds we have defined for the animation.

$D = D\times 5D = 8153.80$

- We now have to set the keyframes for the wheels. Go to keyframe 0. (As described above, the wheels of the vehicle should be individual objects, each parented to the same null object that the vehicle body is parented to.) Select a wheel. The rotational value of each wheel at keyframe 0 is 0.00 degrees. Please note that we are *not* setting the starting position of the wheel at keyframe 1, because frame 1 must have a pitch value greater than 0. Now go to frame 150 (5 seconds later). Set the rotational pitch to 8153.80 (Figure 4). Set keyframe 150 with this value for each wheel. Done. The wheels will now travel at 30 mph for five seconds.

In the Hummer scene in Figure 1, I moved the Hummer at 15 mph for seven seconds. For those of you who have the *LWPRO* disk, load the scene and check it out. There's much more going on than just a Hummer driving across a desert. Also, for those of you who are interested, the scene rendered at an average of 1 hour, 45 minutes per frame on my Amiga 2000 with a 33 MHz GVP '040.


Scene
Objects
Surfaces
Images
Lights
Camera
Effects
Record
Options
SN
Switcher
View
XY
XZ
ZY
Perspective
Light
Camera
Edit
View
Object
Bone
Mouse
Move
Rotate
Size
Stretch
Move Pivot Pt
H
P
B
Reset
Center
Numeric Input
Motion Graph
Preview
Render
Key Frame
Previous
Next
Spline Controls
Create Key
Delete Key
Parent
Goal
OK
Cancel

Object Direction
Heading 0.0
Pitch 8153.80
Bank 0.0
Selected Item Hummer_Left_Wheel
Current Frame 0
Grid: 5.0
Direction
H: 0.00°
P: 0.00°
B: 0.00°

Figure 4: Set the pitch of each wheel to the value D = [(M/C x 360) x length]. Each wheel will then rotate this number of degrees throughout the animation.

Tracks in the Sand

I would now like to touch upon one interesting effect that I implemented in the Full Metal Hummer scene. Many people have commented on it. If you render this scene (or see it on our demo), you'll notice that the tires are leaving tracks in the sand. I've been asked many times to explain how this was done. Well, it's really quite easy.

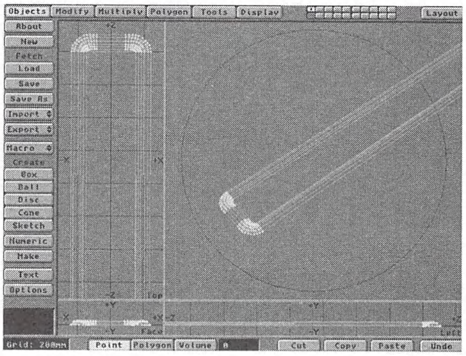

Objects
Modify
Multiply
Polygon
Tools
Display
Layout
About
New
Fetch
Load
Save
Save As
Import
Export
Macro
Create
Box
Ball
Disc
Cone
Sketch
Numeric
Make
Text
Options
Grid: 200mm
Point
Polygon
Volume
Cut
Copy
Paste
Undo

Figure 5: The tread-track object. The raised bump around the edges creates the effect of movement through soft sand.

The track objects are relatively simple. They consist of a surface polygon the tread pattern is mapped to and a raised "bump" that was extruded around the edges of the tread polygon (Figure 5). This was done to create the effect of the Hummer digging into the soft sand as it drove across the desert.

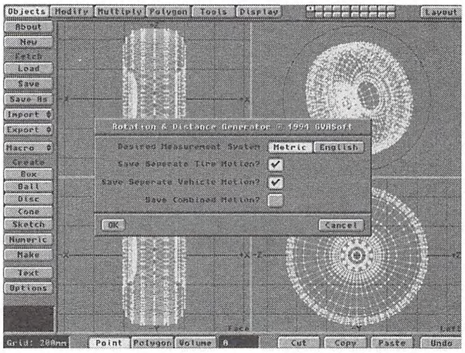
These track objects are parented to the null object, and positioned underneath and a little in front of the front tires. The raised edges have Fractal Noise diffusion and a bump map assigned to them. The tread tracks were created by applying a diffusion, transparency and bump map with a tread image I created in ToasterPaint.

- To create the effect that the Hummer's wheels are leaving tracks in the sand, simply set each texture map to world coordinates. As the track object moves with the Hummer, it will appear to be leaving tread marks in the sand.

Rotation and Distance Generator Macro

Scott Wheeler (my partner) and I developed a macro included on the *LWPRO* disk that easily computes distance and wheel rotation for a given velocity (Figure 6). This macro performs a variety of functions. It allows the user to save three individual motion files:

1. Wheel Rotation Motion File: This selection will cause the macro to save the rotation information computed from the user's input. Individual rotation files can be created for 5 mph, 10 mph, 27.8 mph, etc. Then the wheel can be parented to a vehicle moving at


Objects
Modify
Multiply
Polygon
Tools
Display
Layout
About
New
Fetch
Load
Save
Save As
Import
Export
Macro
Create
Box
Ball
Disc
Cone
Sketch
Numeric
Make
Text
Options
Grid: 200mm
Point
Polygon
Volume
Cut
Copy
Paste
Undo
OK
Cancel

Rotation & Distance Generator © 1994 GVASoft
Desired Measurement System Metric English
Save Seperate Tire Motion? ☑
Save Seperate Vehicle Motion? ☑
Save Combined Motion? ☐

Figure 6: The Rotation and Distance Generator macro makes it easy to move and rotate a vehicle or wheel at any speed.

see Land Vehicle Movement, page 16

Dead in Space

Creating Flares and Explosions for a *B5* Ambush

by Mojo

One of the shots I'm most proud of this season on *Babylon 5* is from the episode "Revelations," in which a heavy Narn Cruiser is ambushed by a mysterious attacker. It exits hyperspace into an evil-looking sector of the galaxy and immediately has a hole punched through it from underneath by the enemy's powerful energy weapon.

The scene turned out to be one of my favorites and, much to my surprise, the producers liked it so much they decided to use it during the opening credits every week. Since this shot has such high visibility, I thought I'd share some of the details of its creation and prove that anybody with LightWave and a few hours to kill can make millions of people say "Wow!" every week.

Space, the First Frontier

One of the distinguishing characteristics of *Babylon 5's* space scenes is their strong use of celestial phenomena, such as colorful nebulas and phosphorescent whirlpools. Not only do such backdrops look nice, but they allow us to light scenes darkly, with shadowed objects still visible in silhouette. In addition, they often provide visible sources for fill light and provide a refreshing break from the overused norm of black sky and white stars. When the script calls for a new area of space to be visited, we spend just as much time making the background look right as we do the action.

"Revelations" called for the Narn cruiser to visit enemy territory. These bad guys are evil and nasty, so I wanted to design an area of space that reflected this—a place that you could tell was dangerous simply by looking at it.

Since TV production has such a hectic schedule, there is never time to create every element from scratch. A while back, NASA sent us some striking images taken by the repaired Hubble space telescope, and we've incorporated these into the show when we can to provide a little touch of "real" science.

One of these images, of a fiery orange-red gas cloud, looked particularly hellish and seemed as if it had the right stuff to be the basis of "evil" space.

After spending a little time in Photoshop, I had the image cropped and processed to the point where it looked good. Some blurring was necessary to remove some of the low-resolution artifacting, but in this particular image, extreme sharpness wasn't necessary. What *was* important was making sure the image was NTSC-safe. Reds and oranges are video's worst offenders, so I brought the colors to well below the safe level; even when technically safe, reds can bleed a lot if they're too saturated. I also didn't want the colors to look too overstated, or else they would have detracted from the explosion—the focal point of the shot.

The image (1300x700) was mapped onto a curved polygon and placed behind everything else, even the stars (see the August '94 *LWPRO* for more information on nebulas). I had to be careful not to pan the camera too far, or else it would see the edge of the nebula. (The image didn't taper off at the sides and there was a limit to how far I could stretch it before it looked bad.)

Since this scene was described in the script as being at the rim of the galaxy, I added a subtle ring of stars across the middle of the screen, all with **Particle Size** set to Small. The object was also dissolved out 50 percent to avoid sharp flickering (very bright, small particles against a dark background would alias and shimmer like crazy).

Once I was happy with the setting for my shot, it was time to get into the thick of it and blow up a perfectly good ship.

Ouch! That Smarts!

The script had the cruiser getting hit and exploding in one blow, but I decided to break it down into two stages in order to build a little bit of emotional impact: the first hit would make it clear that the Narns didn't stand a chance and the second would finally destroy it (after we had begun to feel sorry for them).

I wanted that first hit to be really violent and look as if a hole was literally "punched" into the ship. As the beam hit the bottom of the cruiser, there was a momentary pause to suggest a power buildup before it burst through the top. I felt it should be as powerful and as devastating as possible when it finally broke through—like a bullet through a balsa wood factory.

After it was hit, I thought it should "sink" in front of the camera, implying that this one blow had really crippled the ship. Although the actual physics of this is wrong (in space a damaged ship would certainly not sink), I felt the depressing feeling viewers would get seeing this gutted ship fall past the camera far outweighed the incorrect mechanics of it.

Now I just had to figure out how to accomplish everything!

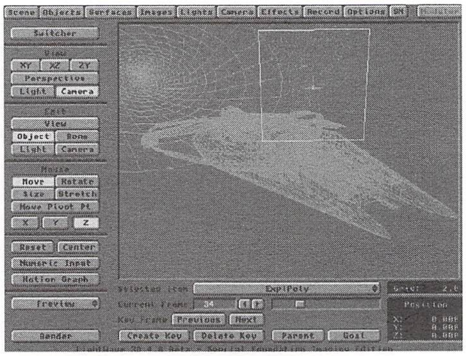


Figure 1: This is what it looked like in layout. Notice the hulking square explosion polygon. It's facing the camera and not orienterd in the direction of the ship.

Down and Dirty

The first step was to use Boolean functions and cut a nice chunk out of the Narn cruiser model. This new version of the ship was parented to the original and cross-dissolved with it while the lens flares of the initial explosion covered the area in question. (This is also why the far side of the ship was chosen to get hit.) Keep in mind that if you're going to try this, make sure you set the dissolve envelope splines to **Linear.** Otherwise, even in a one-frame transition, you'll see a shimmering between objects.

As the beam slices through the top, there are three basic elements that create the impact: lens flares, particles and a real explosion mapped onto a polygon.

Ah, lens flares...there are no less than 120 of those little suckers in this scene file. That's what was needed to create the "burning hole" effect of fire surrounding the area cut from the ship as it sinks past the camera. (The details of this particular effect were explained in the November '94 issue of *LWPRO*.) Of these, only a few were actually needed for the initial impact flare-up. They were ramped between 100 and 300 percent for a few frames to highlight the beginning of the blast, then brought down to a manageable level. (The damaged version of the Narn ship was dissolved in during this period.) These flares also had a matching light intensity envelope, so the orange light from the explosion would be cast on the ship's surface. A falloff was used to keep the light from scattering across the entire object, helping to convey the sense that this is a very large vessel.

Just as the first flares reach a crescendo, a massive outpour of sparks rises from the point of impact. Instead of using a spherical particle cloud, I created a plume of points that stretched vertically, once again helping to emphasize the upward force of the explosion. The object was made by stretching, tapering and using the magnet tool on a macro-generated point distribution of 600 particles. I then saved this object as a source and dragged conglomerations of particles toward the edges and upward, saving this as a morph target. An enveloped morph of this object over 30 or so frames would create the illusion of the sparks moving at various rates.

The particles are then sized to zero, parented to the ship and sized larger at a very quick pace in time with the explosion (along with the simultaneous morph). Several layers of sparks were added to create extra density and simply make it look more exciting.

The finishing touch was the real explosion on a polygon. As good as lens flares and sparks are, there's simply nothing quite like a good old-fashioned fireball. A commonly available CD-ROM called **Pyromania** features frame sequences of several explosions, one of which is a sort of nuclear-style blast heading upward. We don't normally use them on *B5* due to the limitations of polygon mapping, but in this case everything seemed just right to make use of them. In addition, because we don't use such explosions very often, they would create the extra impact I was looking for.

Creating this effect is just about as simple as it sounds. I put a flat polygon on top of the ship in the area I wanted the plume to come from and planar-mapped the image sequence of the explosion. I gave the sequence a frame offset of -12 so it would begin on frame 12 of the animation and made sure the polygon was dissolved out until then.

One thing to keep in mind when using polygon-mapped sequences is dimensionality. If you're not careful, the 2D element in your 3D world can be detected. This usually happens if you move at too extreme an angle to the polygon and its perspective begins to shift. As a result, the image on the polygon will start to "flatten" out, much the same way an image on a piece of paper does if you look at it at an angle. Whenever you use an image on a polygon in this fashion, make sure it faces the camera as squarely as possible at all times, even if it means rotating it during the course of your animation. A good rule of thumb might be to avoid using this trick when you have extreme panning camera moves. In this case, the move was subtle and the polygon was always directly facing the camera, providing the perfect opportunity for this technique.

The main problem with using explosion sequences like this are the polygon edges. When most of these explosions were filmed for the CD, the flames went off the edge of the screen. When mapped onto a polygon, this means that once your explosion reaches that point, it will simply vanish off a hard edge. This would be a dead giveaway of the trick and look pretty awful to boot. In this case, the gas cloud reaches fairly high before it gets to the edge of the frame, so I simply had to make sure the polygon was dissolved out before it got that far. This worked out well, since I really only wanted the fireball for the beginning of the explosion to highlight the impact. After that was accomplished, the lens flares and particles did a more than adequate job of completing the effect.

Finishing Touches

It's always the little things that make a good shot work well. One of my favorite elements in this case is the blue vortex that disappears behind the cruiser during the shot. The idea is that the ship is attacked just as it emerges from the hyperspace vortex, so I added it to the shot for several reasons.

First, I think it looks nice. As a rule, it's important to have more than one color in an image if you want any one color to stand out. In this case, it's the blue of the vortex that makes the reds and oranges of the nebula and explosion so vibrant—it gives your eye something else to reference.

There was also an emotional consideration.

I thought it would make the Narn's situation seem even more hopeless if you saw their only form of escape close behind them while being attacked.

It drives home the fact that they are all alone in a very bad neighborhood, and also presented an opportunity for some neat lighting. (If you look closely you can see the blue light from the vortex play across the back of the cruiser.)

In the end, a combination of several simple techniques makes this shot work extremely well. However, the most important element of this scene is its design. A lot of thought went into *why* to make things look a certain way and not just *how*. Consider this when creating your own scenes and you may find that a little attention to *why* may save you a lot of *how*.

Mojo has been in therapy, trying to resolve his emotional attachment to explosions. He is making progress and has recently conceded his Superior Elvis Fan complex to Colin Cunningham, who obviously needed something to turn to after working on Robocop: The Series.

Editor's Message

continued from page 3

Also, just recently added as a mailing list is the LightWave plug-in mailing list that is updated by Stuart Ferguson. All the latest info about plug-in support can be found here. You can subscribe to this list by sending a subscribe note (like above) to listserv@netcom.com. The name of the list is lwplugin-l.

One of the reasons that I've been thinking about the net is a discussion in the LightWave newsgroup about the CGI work in *Star Trek: Voyager*. I was surprised by the comments by some of the people saying how the CGI work was very obvious and they didn't understand why there was even an attempt to use CGI when they had the models in the first place. One person went on to say that the obvious CGI shots were the ones where the *Voyager* was tiny in the distance and looked very flat-lit. I, of course, took great pleasure in explaining that if it was so obvious, why did they pick the wrong shots to label CGI? None of the CGI shots of the *Voyager* were little in the background (unless they started off up close and flew away).

As a matter of fact, few people were able to pick out all of the CGI shots. I guess that means we did our job correctly. And the other thing it means is that it's becoming increasingly difficult to spot CGI. I think you'd be surprised at the large amount of CGI work that is being used every single day in the visual medium. Next time you're watching TV, take a good look. Don't just zone out—watch for techniques and styles. Pay attention to lighting, camera angles and storytelling. And then try to reproduce some of the more interesting things you see. You'll be a better animator for it, and chances are you can write an article for *LWPRO* about it!

Speaking of *LWPRO* articles, if you tried creating hair using the macro mentioned in last month's "Fur and Hair" article, chances are you were a bit confused (or angry!). The macro was not printed in the issue, and because of space constraints, it is too long to fit (you probably wouldn't want to type it anyway!). However, it is included on this month's *LWPRO* disk, and for those of you that do not subscribe to the disk, I will make it available to you if you e-mail me at jgross@netcom.com (there's that Internet again!). Or you can call Avid Media Group at (800)322-2843 to have a copy mailed or faxed to you.

One final thing: if you are traveling to Las Vegas for this year's NAB show, make sure to visit NAB Multimedia World at the Las Vegas Hilton. Avid Media Group will sponsor the *Video Toaster User* Pavilion, booth #117, with a large number of vendors present, such as NewTek and Carrera Computers. NewTek will also be on the main floor. The conference runs April 9-13 with the exhibitions running April 10-13. Hope to see you there!

You Bonehead!

Part I: A Beginner's Look at Bones

by Dan Ablan

Ahh, yes. Bones. By now, most of us know what they can do. We know you can make inanimate objects come to life. We know that Bones can make solid objects bend and deform. But not many people use Bones on a regular basis, because they're either too tedious to set up, or when you do, weird things happen to your object. Well, believe it or not, they are not nearly as complicated as you may think.

If you've worked on any other 3D platform, you'll see that only a few, like LightWave, have Bones. Bones are an advanced form of free-form deformation that enable the animator to create fluid character animation. You can even make smooth-flowing curtains with just a few Bones. Bones will also allow you to simply deform an object, and in a sense, model in LightWave's Layout. It's good to think of LightWave's Bones as handles. You probably already know all of this, so let's move on.

I thought I'd put together a tutorial from a fun animation I did when I first got LightWave 3.0 and Bones. Arnie Boedecker and I have a series of Bone tutorials we're putting together for *LWPRO*, based on a big project just completed. There were quite a few tricks we picked up when doing those animations (like those flowing curtains I just mentioned), so keep your eyes on future issues of *LWPRO*.

The scene I'm talking about in this article is probably one of the most common animations people have tried using Bones, thanks to, you guessed it, the Listerine ads. Those sensational animations were produced by a company called PIXAR, but there is no reason you can't put together an animation like that at home using LightWave and Bones. Just so no one is misled, we'll call our product Blisterine.

Proper Construction

- The first thing to do is build your bottle. This type of object is pretty easy to build using **Lathe**, so I won't go into details. The way I built mine was by going to the store, buying a bottle of Listerine, and placing it on top of my computer. Then I built it. A more mathematical approach can be used by measuring. Plus, you can grab a frame of the bottle, then trace over the image in Modeler.
- For the label, I used Modeler's **S Drill**, **Stencil** feature (**Tools** menu) to stencil in the label area on the bottle. To create the label image, I found the ToasterFont that looked as much like the original label's fonts as possible. Then I made sure that the spacing was the same, and the size of each section of words. What the words read, however, is not at all what's on my bottle. I composed black letters on white in CG, then imported that frame to ToasterPaint and drew the border. I saved this as a full-size image, so that in the animation, when the bottle jumps forward and lands in front of the camera, you can clearly read the label. It takes up more RAM in LightWave, but it's well worth it when everything is sharp and clear in the final piece. For this tutorial, you can use the pop can supplied in the Toaster's objects directory to save time.

With Bones, anything you want to deform needs to be made up of many polygons. For instance, if you took a storm window off of your back door and tried to bend it, it wouldn't bend at all, or it would crack. But, if you took a screen off of that door and tried to bend it, it would bend very smoothly and easily, right? That's because the screen is made up of many segments. Think of your objects the same way.

In Modeler, to create many segments, use **Subdiv** (**Polygon** menu). You first need to **Triple** (**Polygon** panel) anything before you subdivide. [Editor's note: With Modeler version 3.5, you can subdivide three-sided *or* four-sided polygons.]

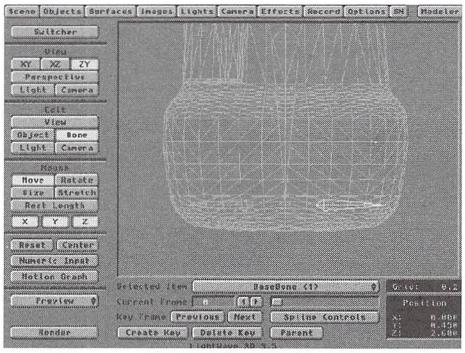


Figure 1: The first base Bone in place as an anchor.

You should also make sure to check for non-planar polygons (Polygon **Stats-Display** menu) and triple any that you find. If your whole object is composed of triangles, you will have no non-planars. This is best for objects that will be deformed in Layout.

- Once this is done, **Export** your bottle to Layout, saving it as bottle.sbdv or soemthing similar. Remember that Bones affect the points of an object, not the polygons, so when you start moving your Bones, don't be confused by the bounding box of your object staying in place.

The Setup

One thing to remember is that Bones are saved with the scene file, not the object. Bones have an unlimited influence on an object. If you don't limit the influence with the **Limited Range** setting in the **Object Skeleton** sub-panel, your entire object will be influenced. The more common way of taking care of the influence is by placing other Bones around the object.

On my Blisterine bottle, I've placed four Bones in the base, renaming them base Bones. Those base Bones don't move because they will act as anchors for the object. Because I'm going to move and rotate the other Bones, the anchors hold the bottle on the ground. The result is a bottle that bends and twists. If those base Bones weren't anchoring the bottle, the entire object would move.

Under the **Objects** panel in Layout, you'll see another button labeled **Object Skeleton**. This is where Bones are loaded, cleared and renamed for your scene. In order to place Bones to a specific object, that object must be selected before you go to the skeleton panel.

- Click **Add Bone**. Rename that Bone "basebone." Return to Layout. You'll see what looks like a squared necktie. That's your bone (Figure 1). This Bone has no effect on your object yet. Select **Continue** and return to the Layout window.
- The next thing to do is choose the object, select **Bone** and choose **Rest Length** from the mouse function control area in Layout. Do not confuse Rest Length with Size. The size will actually change the shape of the object, whereas the rest length determines the amount of influence the Bone has.

The larger the Bone rest length, the more influence it will have on your object. Since I'm placing this first Bone as an anchor, it's sized to half the length of the base of my object. When I add the next three Bones, they will be the same size as well. From this point, you can repeat the process, or if you are working with LightWave 3.5, the **Add Child Bone** feature in the skeleton panel can be used.

This function will clone the Bone and parent it to the currently selected Bone, so you don't have to reset the rest length. **Add Child Bone** is really helpful when putting bones in a snake- or rope-type object.

- Now, rotate the Bones you just added so that you have a base Bone in four places on the base of the bottle—north, south, east and west—and create a keyframe at zero for each. So far, these Bones still have no effect on your object. To make them have influence, press **(r)** to determine the rest position and direction. If you now move or rotate these Bones, they will affect the object. However, we don't want to move these. They're the anchors, remember?
- Go back into the Object Skeleton panel by pushing **(p)** on the keyboard while a Bone is selected. Add another Bone (not a child Bone). Rename it "bottombone." This Bone will be used to control the bottle's lower half. Change the Rest Length and rotate it 90 degrees vertically. Think of this Bone as a leg for our bottle.

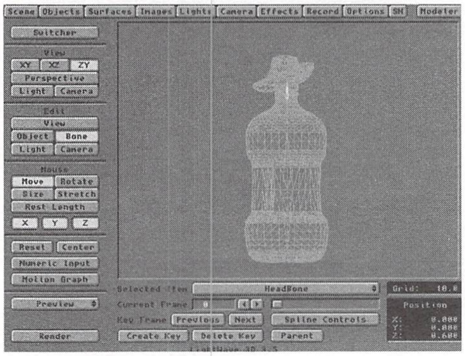


Figure 2: The neck Bone in place.

- Add another Bone and rename it "chestbone." Place this one in the top middle portion of the bottle. Finally, add only one more Bone, renaming it "neckbone," and change the rest length, then rotate it to fit it in the neck area (Figure 2). Once your Bones are in place, type **(r)** on the keyboard and create a keyframe for each bone.

If for some reason you set a Bone wrong, you can deactivate the Bone from the skeleton panel by deselecting **Bone Active.**

- Now, save the scene. Get into the habit of saving the initial Bone setup. That way, if your bone movements get totally out of hand, you can reload the setup scene, and not have to deal with resetting all of those Bones.

Simple Surfacing

I've tried surfacing the Blisterine bottle a number of ways, and recently came across a way to save time by not using traced reflections. You'll definitely need to trace refraction for a glass bottle, though. By loading the Fractal Reflections image and using it as a **Reflection Image** with 10 or 15 percent **Reflectivity**, you'll achieve a nice abstract reflected look, and help your object appear more real. It's a great setting for getting clear plastic to look convincing.

Originally, I made both the bottle and liquid inside it. Each had different surface properties and different refraction indexes. To get true refraction, the light would need to go into the bottle at one setting, pass through another, and leave at yet another setting. Polygon count was high, so just for grins and giggles, I tried something else. I resurfaced the body of the bottle with a different name. I took the liquid out of the bottle and kept one refraction level for the whole thing. Then, I surfaced the body section the color of my liquid, with the neck portion of the bottle fully transparent (see below). In the final animation, the effect was barely noticeable, and saved time rendering because there were fewer polygons and less refraction to calculate. The downfall of this procedure is that the liquid in the neck area won't slosh around.

Bottle Surface Settings

Color	184, 122, 46
Diffusion	90%
Specularity	80
Glossiness	High
Reflectivity	10%
Reflected Image	Fractal Reflections
Transparency	60%
Color Filter	On
Edges	Normal
Smoothing	On
Refractive Index	1.3

Neck Surface Settings

Color	200, 200, 200
Diffusion	80%
Specularity	80%
Glossiness	High
Transparency	100%

The Motions

Now it's time to animate. One thing to learn is to be patient with your movements. Too often, animations are rushed to get a certain amount of movement accomplished within a certain time frame. Work to avoid that. With Bones, timing is everything. The best way to know how to move your character around is to study motions from everyday life. When creating a more cartoonlike character, the movements are exaggerated, so watch how traditional cell animations are drawn and animated.

From here, it's totally up to you how everything will work. For my bottle, I decided to make it look around, then lean back, lurch forward and flip in the air, landing in front of the camera. This became a little tricky, because the Bones bend the object, but keyframes needed to be set for the bottle as well. The chest Bone moves back, making the bottle lean, then it quickly swings forward, and at that point, the bottle starts to flip up in the air. Keyframes for the bottle needs to be set when it starts its jump, in mid-air, when it's upside down, and again when it lands. To add to the whole motion, consider stretching your object on the Y axis, when it lands, while moving the Bone forward quickly, then back, as if the bottle almost loses its balance when landing. The motions for this movement are on this issue's disk with a pop can object ready to animate.

What really helps sell the character, whether it's a bottle, can, glass, or even a square box, is proper keyframing of the motions. To take your animation one step further, consider adding hands, feet or weapons to your object. Since I'm from Chicago, I thought it appropriate that the Blisterine bottle was connected—you know, with the Mob.

Once you've set up the object's movements with Bones, save the scene. Then create the surrounding scene, and simply use the **Load From Scene** (objects panel) feature to import your Bone sequence. I've put my mobster in a scene, but made it black and white to fit the time period. By parenting a machine gun, among other objects, and keyframing that, the illusion is created that the bottle is holding the gun (see color pages).

Remember This

Bones are a very powerful tool. What I've learned in working with them over the past year and a half is patience. The latest project with bones went so well because the entire animation was storyboarded first. We knew what we wanted to accomplish, and when it came time to animate, everything fell into place.

By the way, there is a Ninja and Rambo Blisterine bottle animation in the works.

Dan Ablan is a LightWave animator for AGA in Chicago. He's done work for The Dial Corporation, Kraft Foods, NBC in South Bend, and others. He can be reached at (312) 239-7957 or via Internet at dma@mcs.com.

Beginner's Steps

- Think of Bones as handles to pull and push an object.
- Objects need to be tripled, and sometimes subdivided, to bend properly.
- Rest Length sets the amount of influence of the Bone.
- Sizing a Bone changes the size of the object, not the Bone.
- Use Add Child Bone (LightWave 3.5) to save time.
- Always save the scene once Bones are in place.
- Save scene with Bone movements under a different name.
- Be patient with the amount of movements.
- Stretch the object while moving Bones for added characteristics.

Popular 3D Algorithms

Part II: Ray Tracing and Radiosity

by William Frawley

If you tuned in last month, you'll recall my explication into the nature of the Z-buffer rendering algorithm. If it's a bit hazy or you're just joining us this month, don't worry, we'll trip the light fantastic with a short review to get us primed for the way-cool exploration of the wowities of ray tracing and radiosity techniques. (Sorry 'bout the '60s-like vernacular, dudes, but I never did like introductions. Hang tight, you stiffers, we'll get back to "professionalism" in the next paragraph.) Trust me, if you make it through this algorithm article, you'll never look at that Render button in the same light again.

The Review

As you recall, all objects exist within 3D object space, bound on all sides by clipping planes. Any object or part of an object lying outside of a clipping plane does not enter into the final image. For the most part, the side clipping planes are determined by the viewing angle, and the hither (near) plane usually coincides with the viewscreen. The yon (far) clipping plane may be some arbitrary distance determined by the programmer or may be a virtual backdrop image like that available in LightWave's **Effects** panel. When 3D objects are finally rendered, they are "projected" onto a 2D screen called the viewscreen. This, incidentally, is the final image that is seen.

As we saw last month, the essential ingredient in the Z-buffer rendering technique is the use of frame buffers, not only as the buffer to store the final image, but also as a repository for the distances of the nearest surfaces to the viewscreen. With the help of this additional buffer, the algorithm constructs a virtual picture of the closest object surfaces within the scene, which is then used as a guide in the final shading stage of those surface points visible from the camera view. This shading information is then written to the image buffer. Recall that with this rendering technique, in its purest form anyway, each object is processed individually, negating the necessity of storing all objects in memory simultaneously. Unlike ray tracing, which we'll look at next, this is a great advantage for low-memory systems. Z-buffer rendering is fast and simple, but for more realism in dealing with more complex objects, ray tracing is usually considered.

Ray Tracing

Due to its procedural parallel to light's behavior in the physical world, this is probably the oldest and most popular algorithm responsible for producing some of the most realistic images ever seen. First popularized in the Amiga community with such groundbreaking software for the personal computer as Impulse's Turbo Silver and then Imagine, ray tracing achieves its marvelous realism by mimicking the physics of light rays—including the laws of reflection and refraction—with prisms, lenses, mirrors and other objects. By its very nature, ray tracing also handles optical effects such as shadows, antialiasing and motion blur well.

Unlike the Z-buffer algorithm, ray tracing can be thought of as a random-sampling rendering technique because each pixel can be computed independently. However, all objects within the scene need to be present simultaneously for the procedure.

The general idea for ray tracing is as follows: Each light source emits rays of illumination. These rays travel through space striking and bouncing off of objects, eventually reaching the viewer's eye. Of course, the odds of the algorithm tracing just the right rays that will reach the viewer are enormous, considering the infinite number of rays emanating from each source. Therefore, ray tracing approaches matters from the opposite direction. Here, a singular ray is traced from eye to pixel out into object space, until it encounters an object (if any). From that surface point, which is now the shading point, the light ray is either absorbed or traced back to its many possible component origins via its reflection and transmission to other objects or its direct illumination from other light sources. The resultant shading for the surface point then is a summation of all these possible sources for the original eye ray. This is much more efficient than starting a ray from a light source and hoping it finally reaches the viewscreen. Interestingly, although an object may not be seen directly within the viewing volume, it may be seen as a reflection in another object. Since a ray may then originate outside the clipping planes, it is necessary to construct a **bounding** sphere initialized to some background color so the ray isn't traced forever (Figure 1).

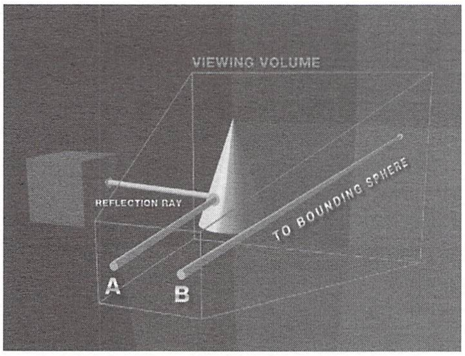


Figure 1: Just because an object lies outside the viewing volume doesn't mean it won't enter into the final image. In ray tracing, an object may contribute light rays via reflection or transmission even though it resides outside of the clipping planes (A). However, if a ray never strikes an object, a virtual bounding sphere ensures that some background color shades the pixel (B).

Basically, ray tracing can be considered a two-step process. First, for each ray passing from your eye through each pixel out into the viewing pyramid, a tree diagramming the interactions of the ray with all the other objects and lights within the scene is constructed. The ray in this case is simply the path the light will take from objects and lights back to the camera. Second, the tree is processed. This is where the shading is computed. The color value of each pixel is determined at this final stage—neglecting any antialiasing and motion blur effects for the moment.

In the construction phase of the ray tree, what happens when a ray strikes an object is determined from the surface properties and the complexity of the scene. If, for example, there is only one opaque, diffusely reflecting object and one light source, a ray traced from the eye striking the surface may follow a path leading directly to that light source. This is called a direct illumination path. It is helpful when tracing a ray to construct an abstract tree diagramming its interactions within the scene (Figure 2). It is these interactions of the ray's path with other objects and lights that will eventually construct the final picture.

Imagine now a more complex scene containing multiple objects with both transparent and reflective surfaces and multiple light sources. The ray strikes the first object, which might be a prism, for example. Because of the prism's surface properties, reflective and transparent, the ray splits into two component rays upon striking the surface: a transmitted ray (because of the transparent property of glass) and a reflected ray (because of the reflective property of glass). Continuing to follow the path of each of these rays might lead to more objects being encountered and subsequently split, depending on the surface properties of these new objects and the angles of the rays leaving the first object. Combine these two types of component rays (reflection and transmission) with the direct illumination rays from each light source and you might get a very complex, multi-level ray tree, depending on the number of lights and objects in the scene (Figure 2). As you might guess, as the tree gets deeper, more calculations are required, and therefore more computer rendering time is needed. It's up to the programmer to determine how extensive the ray tree becomes, but there must be some limit to the ray's path, or rendering of the image might outlast your lifetime.

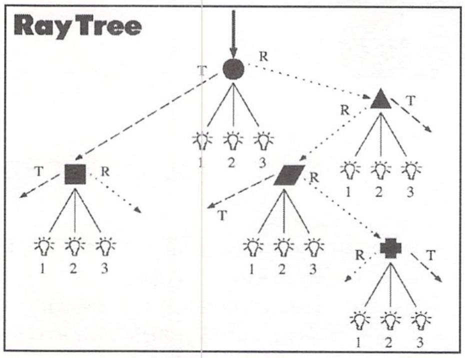


Figure 2: A ray tree diagramming a scene with four objects and three light sources. The construction of a tree helps determine all the possible interactions of a light ray as it travels within the scene. For each pixel, a ray is typically shot from the eye outward until it strikes an object (if any). From there, the surface material deems whether the ray splits into reflection or transmission paths. The direct illumination paths are also calculated.

To process the ray tree, consider all the possible incoming sources of light rays striking the surface point in question. Then sum the entire illumination contributed by these sources, be it direct, reflected or transmitted rays, to find the outgoing light to the appropriate pixel. Therefore, an important part of the calculation for the shading of a point takes into account the object's surface properties and the interaction of these properties with the summation of all the incoming colors and intensities of the various illumination rays. It is important to note that the reflected and transmitted rays that we are dealing with here are specular in nature, not diffuse. Remember that specular reflection is such that the path of light obeys the angle of incidence/reflectance law — one unique path for the ray relative to its origin. We'll explore diffusion effects when we discuss the radiosity preprocessing technique shortly.

Finally, to determine if a shadow contributes to the shading information, each light source is considered for its potential illumination of the surface point. By tracing a ray from each light source to the object, you determine if any other object blocks the illumination ray's path. If so, the point must be in shadow. If not, the total illumination is added into the shading. Because of the mathematics of this algorithm, however, it is possible to include penumbra effects, or partial shadowing, by calculating the proportional light intensities reaching the surface point of partially blocked objects.

Because each pixel can be thought of as a separate entity for processing, all objects must reside in memory at once when building the ray tree. This approach differs from the Z-buffer technique, which deals with one object at a time for each Z-buffer pass. But this is again due to the mathematical nature of ray tracing, which does have the benefit of locating hidden surfaces and knowing immediately which object is the first to be hit by a ray.

Other limitations with basic ray tracing (although each programmer may fine-tune various aspects) include realistic shadows and **caustics**, extremely focused spots of light. There are many ways around this problem, however, such as creative reflection mapping (*LWPRO*, Vol. 1, No. 9, "Beneath the Surface"). Finally, the biggest hurdle for obtaining extremely realistic images lies with the phenomenon of diffuse inter-reflections.

Diffuse Inter-reflections

Diffusion, or the scattering of light based upon a surface's material properties, is a concept you may already be familiar with if you've dealt with surfacing in LightWave. Recall that a more diffuse material scatters light in all directions, whereas a less diffuse surface tends to focus or absorb incoming light rays. Diffuse inter-reflection then is the reflection and transmission of scattered light between different surfaces (no surprise). Unlike specular reflection and transmission, which depend also on the glossiness of the surface, this phenomenon is usually witnessed regardless of the viewing angle.

For instance, imagine the corner of a normally lit room. One wall is red, the other is white. What you would probably notice is that the junction between the walls is a little darker than the flat surface of the walls, and the corner of the room even darker yet. This gradual decrease in light reaching the corner is the result of a falloff of light reflecting and scattering from surface to surface. In fact, at the junction of both the red and white wall, you would notice that the white wall is tinted slightly red and the red wall is a little brighter due to the scattering of light of the white wall. Consequently, the total light reaching each surface is the sum total of both the light source and the diffuse inter-reflections. If the walls are glossy, throw in some specular reflection as well.

```
RADIOSITY BALANCING LOOP PSEUDO-CODE

    Calculate Form-Factors:

    FormFactor.A.B = % based on the orientation of each element to every other element
    FormFactor.A.C = " "
    FormFactor.B.C = " "

Initialization Loop
  {
    Calculate Emittance (if any) of Each Element:

    Emit.A = Intensity (if light source)
    Emit.B = " "
    Emit.C = " "

    Calculate Reflectance (incident light) of Each Element:

    Reflect.A = ? = 0
    Reflect.B = ? = 0
    Reflect.C = ? = 0

    Calculate Radiosity of Each Element:

    Radiosity.A = Emit.A + Reflect.A
    Radiosity.B = Emit.B + Reflect.B
    Radiosity.C = Emit.C + Reflect.C
  }

    First Iteration of Balancing Loop:

Radiosity of one element contributes
to another's incident light

    Reflect.B = Radiosity.A * FormFactor.A.B        A→B
    Radiosity.B = Emit.B + Reflect.B

    Reflect.C = Radiosity.A * FormFactor.A.C        A→C
    Radiosity.C = Emit.C + Reflect.C

    Reflect.A = Radiosity.B * FormFactor.A.B        B→A
    Radiosity.A = Emit.A + Reflect.A

    Reflect.C = Radiosity.B * FormFactor.B.C        B→C
    Radiosity.C = Emit.C + Reflect.C

    Reflect.A = Radiosity.C * FormFactor.A.C        C→A
    Radiosity.A = Emit.A + Reflect.A

    Reflect.B = Radiosity.C * FormFactor.B.C        C→B
    Radiosity.B = Emit.B + Reflect.B

    Second Iteration of Balancing Loop:
    .
    .
    .
```

Figure 3: A pseudo-code example to help show how radiosity is calculated. For simplicity, only three elements (polygons) are considered here.

Diffuse inter-reflections cannot be solved with ray tracing alone. Ray tracing handles the shading of surfaces strictly by calculating the rather straightforward geometries of specular reflection and transmission, taking into account the surface properties of other objects and the color and intensity of lights striking such surfaces. Since diffuse light can originate from an infinite number of paths, it would be time-prohibitive to shoot a large number of rays in all directions just to determine if light is being diffusely reflected from some

see Popular 3D Algorithms, page 17

In the March Issue:

Next month's issue of *LIGHTWAVEPRO* takes a look at LightWave's use in *Star Trek: Generations* and *Star Trek: Voyager*.

Mighty Morphin' Morphing Tricks

by James G. Jones

What is morphing?

Well, this article doesn't have anything to do with 2D image morphing, the type made popular (perhaps too popular) by Michael Jackson's music video "Black and White." And, although morphing is "mighty" powerful, I'm certainly not talking about those annoying adolescents in multi-hued spandex who gallivant around the tube on Saturday morning.

This is about 3D morphing: the ability of LightWave to change the geometry of one object to match the shape of another.

Limitations

There are a few limiting factors. First and foremost is the requirement that the source object and the target object have the same number of points. Trying to morph an eight-point box to a 34,000-point Tyrannosaurus will not work.

Furthermore, with few exceptions, the target object should be modeled directly from the source object. In practical terms, this means you create a source object in Modeler, then bend, stretch, twist, move, resize, rotate, push, pull, dent, drag, kick, mangle and otherwise reshape your original object into its morph target. In some instances, you might have to resort to moving points around individually.

One very important thing to remember is to save the source object before getting out the hammer.

Another limitation is that as the points of the source object move to their new positions in the target object, they move in straight lines. However, there is a way around this that we'll get to in just a moment.

Digression on Theory

Many of us have a difficult time understanding just what is going on with source objects, target objects, metamorph levels and envelopes. If I had to describe the basic principle involved in a succinct manner (and I guess I do), I'd say that you should think of a morph target merely as data that the source object "looks" at to determine what shape to become. If you keep that thought in mind, I think everything else will be a bit more understandable.

How to Set Up a Morph

It's all about percentages, envelopes and who is targeting whom.

Say you have three objects: a bust of Bill Clinton, a mailbox and a back half of a horse. They were all modeled from the same object and have the same number of points.

This is obviously a hypothetical situation, so just pretend you're doing the following steps:

- Load the three objects into Layout. Set the mailbox and horse's butt to 100% **Object Dissolve** (**Objects** panel). You could just move them out of camera view, but I like to use Dissolve because it's right there in the objects panel.
- Set the **Metamorph Target** for Clinton to be the mailbox, and set the **Metamorph Level** to 100%.
- Set the **Metamorph Target** for the mailbox to be the horse's hiney, and set the **Metamorph Level** to 100%.

Now close the Objects panel. What do you see? Yes, that's right: Bill Clinton is a horse's ass. (I've always wanted to say that.)

But why? OK, here's what's happening: the mailbox is 100% morphed into the shape of the horse's behind. The bust of Clinton is 100% morphed into the shape of the mailbox. But since the mailbox is now shaped like the equine posterior, that is the shape to which William is morphed. Get it? If not, read the preceding paragraph again. Slowly.

This is called a "chain" of morph targets. In other words, object 1 is morphed to object 2. Object 2 is morphed to object 3. Object 3 is morphed to object 4. (Object n is morphed to object n+1, for you math heads.) This can go on for quite a spell—up to 16 times for any one source object, according to the manual. Rumor has it, though, that you can actually have far more than 16. I, for one, am not about to set up a 97-object chain of morphs just to find out the real maximum. If you have excess time on your hands, feel free. [Editor's note: For those of you without excess time, the limit is 40!]

To control the morphs over time, you use envelopes. For example, you'd set up an envelope for the Clinton bust object with a keyframe at frame 0 equal to 0%, and a keyframe at frame 15 equal to 100%.

Then you'd create an envelope for the mailbox that goes from 0% at frame 0 to 0% at frame 15 to 100% at frame 30.

The resulting animation would show Clinton's head at frame 0, changing to the mailbox at frame 15, changing to the horse's hindquarters at frame 30.

Morphing Real Objects

Enough theory...here's a slightly more complex (and less political) example that you might actually find useful.

How about morphing a flat plane to a sphere?

Having an animation where a flat plane changes to a sphere is one of those things that sounds easy to do at first, but turns out to be a bit tricky in reality. Here are a couple of approaches I've worked out: one solves a problem with the Wrap-To-Sphere macro in Modeler, is very simple and gives OK results. The other is a bit more complex but looks much better.

Method One

- In Modeler, use **Box** (**Objects** menu) to make a flat plane that is twice as wide as it is tall. For this example, 2 meters wide (along the X axis), 1 meter tall (along the Y axis), with no depth (along the Z axis). Be exact. Use **Numeric** (n) to enter the pre-

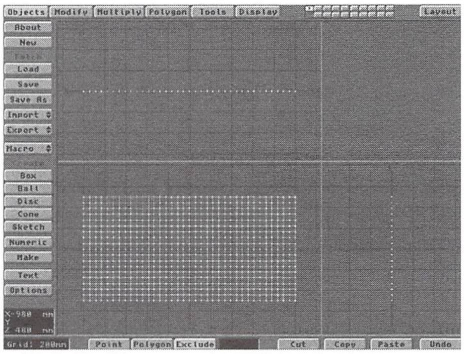


Figure 1: Make a flat plane in Modeler, 36x18 subdivisions.

cise values. Enter 36 subdivisions along the X axis and 18 subdivisions along the Y axis. You should end up with a flat plane that looks just like Figure 1. Use the Center macro to center the plane if you built it off-center.

- You will notice that the polygon's surface normals will be facing toward you (toward the negative Z axis). **Flip** them all (**Polygon** menu or f) so they are facing away from you (toward the positive Z axis). This will make sense in a moment, trust me.
- Save this object as "FlatPlane."
- Now go to the **Macro** button and choose "Wrap-To-Sphere." When the requester appears, click on the **All** button and set the inner radius to 0.5 meters. Click on **OK**.

You will now have a nice-looking sphere, and you will notice that the polygons are facing the way they should—outward.

- Save this object as "Sphere."
- Now load these two objects into Layout and set up a morph from the FlatPlane to the Sphere over a period of 30 frames. Don't forget to set the Target object (Sphere) to 100% dissolve. Make a wireframe preview and the first problem you'll notice is that the flat plane turns inside-out on its way to the sphere shape.

This is not good. However, you could always run it backward and do an animation of an imploding grapefruit.

By the way, I have no idea why this macro flips the points around like this. I just use macros, I don't understand them.

- To solve this problem, go back to Modeler and **Import** the FlatPlane.
- Use the **Rotate** tool (**Modify** menu or y) to turn it 180 degrees on the Y axis. Now the polygons will (surprise) be facing the right direction—toward the camera.
- Save it again and **Export** it back into Layout.

Well, the flat plane no longer turns inside out and the morph looks fairly good. However, look closely at the corners of the flat plane as it changes to the sphere. They fold back on themselves in a rather unpleasant manner. This is what prompted me to come up with...

Method Two

This approach uses the previous two objects and creates a third object, a cylinder, as an intermediary between the flat plane and the sphere.

- Load the FlatPlane object into Modeler.
- Click on the **Volume** button. It should say **Exclude**. If not, click on it again.
- Using the left mouse button, draw out a selection box in the Face view around the right half of the plane. The left edge of the selection box should be exactly at the center line (where X = 0). Now, when you go to bend this object, it will only affect the right half.

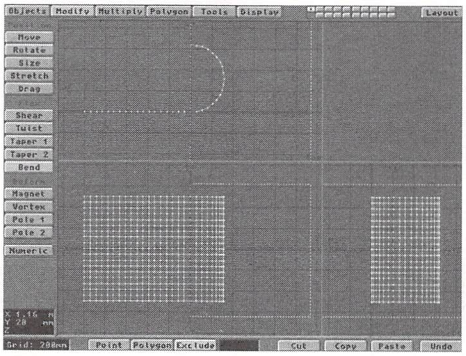


Figure 2: Use the Bend Tool to bend the right half of the plane back 180 degrees.

- Activate the **Bend** tool (**Modify** menu) and **Numeric**. In the requester, enter the following values:

Axis	X
Range	Automatic
Sense	positive (+)
Angle	180
Direction	90
Center	0, 0, 0

- Click on Apply.

You should see something like Figure 2. If the plane's end points do not bend back far enough where they are located at X=0, click **Undo**, select all of the middle points (where X=0) and use the **Set Val** tool (**Tools** menu) to set all of the middle points to X=0. Then you can repeat the above step. Sometimes, after centering an object, the middle points may be located at X=-0. Those points will not be affected in a volume that rests on them.

- Now turn off **Bend**, and move the volume selection box to surround the left half of the flat plane. This time, the right edge of the selection box should be at the center line. Choose **Bend** and **Numeric**, and use the same values as before, except change Sense to negative (-) and **Direction** to -90.
- Click on Apply.

You should have a cylinder. However, you'll notice that it's not centered.

- Use the Center macro to center the object at 0, 0, 0.
- Save this object as "Cylinder."

Setting Up the Morph

- Load all three objects—"FlatPlane," "Cylinder" and "Sphere"—into Layout.
- For Cylinder and Sphere, adjust the Object Dissolve to 100%.
- Set the **Metamorph Target** for Cylinder to Sphere, and the **Metamorph Target** for FlatPlane to Cylinder.
- Create a metamorph envelope (**E** button) for FlatPlane with keyframes at 0 and 30. Set the tension for keyframe 0 to -1 and the tension for keyframe 30 to +1.
- Make a similar Metamorph Envelope for Cylinder, but set the tension for keyframe 0 to +1 and the tension for keyframe 30 to -1.

Notice that the morph envelopes overlap. While the plane is morphing to the cylinder, the cylinder is morphing to the sphere.

- Make another wireframe preview and see what you think.

Because of the overlapping morph envelopes and the tension settings, the cylinder object is exerting more influence at the beginning of the morph, which makes for a more aesthetic change to a spherical shape. Note also that some of the points of the flat plane no longer travel in a straight line to their destination.

Some Morph Tips

- Any texture map that you have applied to the flat plane goes along for the ride when you morph it into the sphere.
- If you need to see the back of the sphere after the morph, do a one-frame dissolve to a sphere that has the points where the edges merged. Otherwise you'll see a seam.
- If you want to see the sphere morph down into a flat plane, simply set up your morphs so your plane object starts out 100% morphed and then goes down to 0%, or morph the sphere into the flat plane.

Morphing as Movement

Morphing is about movement. The points move. The polygons defined by those points move.

Why not use a morph to move a whole object?

Why would you want to, considering that LightWave allows you to move objects directly in Layout and set keyframes and all that?

Well, say your animation needed a mosquito. A rabid, hyperactive, directionally challenged mosquito, to be precise.

- Create a small (maybe 500 mm on each side) cube in Modeler.
- Move it (t) off to the lower righthand corner of the Face view.
- Save this object as "Box1."
- Move it again, this time over to the lower lefthand corner.
- Save as "Box2."

see Mighty Morphin' Morphing Tricks, page 16

What's on the Disk?:

This month's disk includes scene files and objects from Joe Dox' Full Metal Hummer scene, a sample bones scene from Dan Ablan, and GVA's rotation and direction macro. Also included from last month's *LWPRO* is Gonzalo Garramuno's Hair Creation macro and Dan Ablan's Wood Scene.

Reader Speak

Explaining Configuration Files

by John Gross

It's been awhile since we've been able to include a Reader Speak column and the mail has been building, so let's respond to some of it this month.

Q: I know there are ways to customize LightWave and Modeler. Could you talk about the config files and how to go about changing them?

Paul Ortega
Lubbock, TX

A: Indeed, you can customize some features in both LightWave and Modeler by modifying some ASCII configuration files. Configuration files for LightWave and Modeler are found in the Toaster/3D directory and are called LW-config (LightWave) and MOD-config (Modeler). Two full sample config files are listed below and then reprinted with explanations and tips.

LightWave Config File

```
LWCO
0
FileReqPreset1 DF0 DF0:
FileReqPreset2 BOX BH0:
FileReqPreset3 WORK DH1:Toaster/3D
FileReqPreset4 EXTRA DH2:Toaster/3D
ScenesDirectory DH1:Toaster/3D/Scenes
ObjectsDirectory DH1:Toaster/3D/Objects
HierarchiesDirectory DH1:Toaster/3D/Scenes
SurfacesDirectory DH1:Toaster/3D/Surfaces
ImagesDirectory DH1:Toaster/3D/Images
OutputDirectory DH1:Toaster/3D/Images
FramestoresDirectory DH1:Toaster/3D/Images
MotionsDirectory DH1:Toaster/3D/Motions
EnvelopesDirectory DH1:Toaster/3D/Envelopes
PreviewsDirectory DH2:Toaster/3D/Previews
StatusFilename (none)
DefaultTension 0.000000
DefaultSegmentMemory 2200000
DefaultZoomFactor 3.200000
DefaultOverlay 0
FrameEndBeep 1
RenderDisplayDevice 4
RecordSetup1 (none)
RecordSetup2 (none)
RecordCommand (none)
RecordDelay 0.000000
FirstFrameDelay 0.000000
DefaultLayoutGrid 8
AutoKeyAdjust 0
ExpertMode 0
```

Modeler Config File

```
MDOP
0
FlatnessLimit 0.500000
QuadTriMode 1
TwoSided 0
CurveDivision 1
UnitSystem 0
FontsDirectory /ToasterFonts
MotionsDirectory Motions
MacrosDirectory DH2:LWM
MacroListsDirectory DH2:Toaster/3D
MacroList DH2:Toaster/3D/JGMacros
StartupMacro
KeyMacroF1 DH2:LWM/Text.lwm
KeyMacroF2 DH2:LWM/Router.lwm
KeyMacroF3 DH2:LWM/PointSpread.lwm
KeyMacroF4 DH2:LWM/LightSwarm.lwm
KeyMacroF5 DH2:LWM/CutFaces.lwm
KeyMacroF6 DH2:LWM/NextEmpty.lwm
KeyMacroF7 DH2:LWM/NearBG.lwm
KeyMacroF8 DH2:LWM/PointCenter.lwm
KeyMacroF9 DH2:LWM/Center1D.lwm
KeyMacroF10 DH2:LWM/Center.lwm
ColorInterface 1
ScreenModeID 0
```

Both config files have several things in common. Some defaults are automatically written whenever you exit the program, like most of the Modeler options, and some you need to type in yourself using any word processor or text editor, such as most of the LightWave options. Both files list an option, all as one word, then a space, and then the default (whether user-defined or written by the program). By the way, both of these files are for the shipping (3.5) version of LightWave and may be different from those for your version.

Let's break them down. All explanations and comments will appear in italics.

LW-config

```
LWCO
0
```

These first two lines are file identifiers and should not be modified.

```
FileReqPreset1 DF0 DF0:
FileReqPreset2 BOX BH0:
FileReqPreset3 WORK DH1:Toaster/3D
FileReqPreset4 EXTRA DH2:Toaster/3D
```

The FileReqPresets allow you to change the four drive buttons listed at the top of all LightWave requesters. The first word after the space is the name you wish to appear on the button, while the second word (or group of words) is the full path you wish to appear when you click on the button. These four are not written by LightWave. You must assign them or LightWave will use its defaults.

```
ScenesDirectory DH1:Toaster/3D/Scenes
```

This is the directory you wish LightWave to look at when you click on Load Scene. It and the following default directories are not written by LightWave, but instead must be entered by the user if you wish to use directories other than the programs defaults.

```
ObjectsDirectory DH1:Toaster/3D/Objects
```

This is the directory you wish LightWave to look at when loading Objects.

```
HierarchiesDirectory DH1:Toaster/3D/Scenes
```

This is the directory LightWave looks for when using Load From Scene.

```
SurfacesDirectory DH1:Toaster/3D/Surfaces
```

The directory LightWave first looks to when you choose to load or save a surface.

```
Images Directory DH1:Toaster/3D/Images
```

This is the directory LightWave looks at when you choose Load Image or Load Sequence.

```
OutputDirectory DH1:Toaster/3D/Images
```

This is the default directory that LightWave looks for when you choose ***Save RGB*** *or* ***Save Alpha.***

```
FramestoresDirectory DH1:Toaster/3D/Images
```

The default directory for loading or saving framestores.

MotionsDirectory DH1:Toaster/3D/Motions

The default directory for loading or saving motions while in the motion graph panels.

EnvelopesDirectory DH1:Toaster/3D/Envelopes

This is the default directory for loading or saving envelopes while in an envelope panel.

PreviewsDirectory DH2:Toaster/ 3D/Previews

Finally, this is the default directory you will see when you choose to load or save a wireframe preview.

StatusFilename (none)

This was an option that was used for certain render farm software and has become fairly obsolete with the advent of ScreamerNet.

DefaultTension 0.000000

By changing this value, you can cause any keyframe generated to automatically have a tension value set to this number (between -1 and 1). This can be handy if you do a lot of logos, as logos generally have tension at their start and end frames.

DefaultSegmentMemory 2200000

*This value is used to determine the amount of **Segment Memory** (Camera panel) LightWave will use when rendering images. If you have enough RAM, you may want to set it higher than this default. This value will cause four segments to be generated for a Medium Res image. If you want to be really anal, a value of 8663040 will give you one segment at Medium Res. Whenever you start a new scene, LightWave will use this number.*

DefaultZoomFactor 3.200000

This value is used to determine the equivalent lens setting of LightWave's camera whenever a new scene is started. This default creates a 24mm lens (assuming you are using a Film Size of 35mm).

DefaultOverlay 0

This value (0 or 1) will determine if LightWave's Data Overlay option will be on (1) or off (0) by default whenever you begin a new scene. If Data Overlay is on, whenever you save a scene, you will be asked if you want the scene name inserted into the Data Overlay label. (Data Overlay started life as a seaQuest *feature and has been a lifesaver ever since.)*

FrameEndBeep 1

Like Data Overlay, a 0 will default Frame End Beep (Scene panel) off whenever you start LightWave and a 1 will default it on. The state of this option is written whenever you exit LightWave. If you exit with it on, it will be on the next time you start, and vice versa.

RenderDisplayDevice 4

This option is written by LightWave each time you exit using your current setting. This determines what the Render Display will be set to the next time you run LightWave. The current options are None (0), Toaster (1), 6-bit HAM (2), 8-bit HAM (3) and Picasso II (4).

RecordSetup1 (none)
RecordSetup2 (none)
RecordCommand (none)
RecordDelay 0.000000
FirstFrameDelay 0.000000

*The five lines above are written by LightWave upon exit. They refer to the text strings entered for **Record Setup 1, Record Setup 2** and **Record Command**, as well as the **Frame Record Delay** and **Extra First Delay** items found in the Record panel. These are used if you single-frame record direct from LightWave to a video device.*

DefaultLayoutGrid 8

*This option is not written by LightWave, but I think it is supposed to be. This determines the default settings for your **Layout Grid** (Options panel). They range from Off (0) to 16x16 (8).*

AutoKeyAdjust 0

*This line is written by LightWave depending on the state of **Auto Key Adjust** (Options panel). A 0 means the button is off, a 1 means it is turned on. Watch out for this. It is great for composing stills, as it will automatically create a new key whenever you modify an item on a keyframe. It can be devastating if you want to just "try out" a new position. Try using it while showing motion graphs for some fun!*

ExpertMode 0

This feature is no longer implemented, but when it was turned off (0), you were warned that whenever you selected a raytracing button (Camera panel), it would take a long time to render.

MOD-config

MDOP 0

This is the file identifier and shouldn't be changed.

FlatnessLimit 0.500000

*This value and all of the remaining values, with the exception of the last two, are written by Modeler each time you exit the program. Flatness limit is taken from the value you enter in the New Data Options (**Options** under the Objects panel) requester. This is the flatness percentage that Modeler will use when determining if a polygon is non-planar or not.*

QuadTriMode 1

*This value—0, 1 or 2—will determine if **Triangles** (0), **Quadrangles** (1) or **Automatic** polygon generation takes place when you create new objects. If set to Automatic, Modeler will generate triangles whenever it is going to create polygons outside of the flatness limit. Otherwise, it will generate quads. These selections are also found in the Options panel.*

TwoSided 0

*A value of 0 will generate one-sided polygons while a value of 1 will generate two-sided polygons. These correspond to the **One Side** or **Two Sides** buttons in the Options panel.*

CurveDivision 1

*The Options selections of **Coarse** (0), **Medium** (1) or **Fine** (2) will be used to determine how smoothly curves should be calculated when performing operations involving curves (such as lathing a curve or generating PostScript fonts).*

UnitSystem 0

*This option is written from the Display **Options** (Display menu). It will determine what unit of measurement is used the next time Modeler is started: **SI** (0), **Metric** (1) or **English** (2). By the way, SI stands for the International System of Units (from the French "Le Système International d'Unties").*

FontsDirectory /ToasterFonts

This is the directory Modeler will look to when loading PostScript fonts. If you find fonts in a different directory from what appeared, Modeler will use that directory next time.

MotionsDirectory Motions

*The default directory for finding motion files for **Path Extrude** or **Path Clone** (Multiply).*

MacrosDirectory DH2:LWM

This is the default directory for finding macros.

MacroListsDirectory DH2:Toaster/3D

This is the default path for finding macro lists used in the << CONFIGURE LIST >> option in the Macro pop-up menu.

MacroList DH2:Toaster/3D/JGMacros

This is the last used Macro list.

StartupMacro

This is the macro that was last selected to be executed when Modeler was started (<< CONFIGURE LIST >>):

KeyMacroF1 DH2:LWM/Text.lwm
KeyMacroF2 DH2:LWM/Router.lwm
KeyMacroF3 DH2:LWM/PointSpread.lwm
KeyMacroF4 DH2:LWM/LightSwarm.lwm
KeyMacroF5 DH2:LWM/CutFaces.lwm
KeyMacroF6 DH2:LWM/NextEmpty.lwm
KeyMacroF7 DH2:LWM/NearBG.lwm
KeyMacroF8 DH2:LWM/PointCenter.lwm
KeyMacroF9 DH2:LWM/Center1D.lwm
KeyMacroF10 DH2:LWM/Center.lwm

The above 10 options are written into the config file if you selected any keyboard macro shortcuts with the << CONFIGURE KEYS >> option in the Macro pop-up menu.

ColorInterface 1

This option is for the standalone Modeler and determines if Modeler will be started with a four-color interface (0) or an eight-color interface (1) when run by itself. Otherwise it takes on the interface that LightWave is using. This option and the next one are the only two not written by Modeler when you exit the program.

ScreenModeID 0

This number is determined by the ChangeMode program that comes with the standalone version of LightWave and is used to determine the screen resolution of Modeler when it is run by itself.

Land Vehicle Movement

continued from page 5

the same speed. Which brings us to:

2. Vehicle Movement Motion File: This selection will cause the macro to save vehicle movement along a particular axis. This can be used to create motion files for individual vehicle speeds (55 mph, 60 mph, 120 mph, etc.).

3. Combination Movement and Rotation Motion File: This selection will cause the macro to save a motion file containing both vehicle movement and wheel rotation information.

After the types of motion files, if any, have been defined, the user is prompted for wheel radius, speed and the number of frames in the animation. If there is an object in the current layer, the macro will attempt to compute the radius for you. It will then show you the results of the object motion, and prompt you to save the predefined motion files. We hope you find the macro useful.

About the Hummer Scene

I actually modeled the Hummer about two years ago. As a reference, I used a few pictures I saw in *Time* magazine. It was only recently that I went back to it and armed it to the teeth. I added the four barrel machine guns and the side missile launchers earlier this fall. Needless to say, it's one mean machine. But I seriously doubt that a vehicle with this much firepower actually exists, or can exist.

Beginner's Step-by-Step for Land Vehicle Movement:

1. Define vehicle's speed.
2. Define length of animation.
3. Compute vehicle's distance traveled with the formula:

$$M = \cfrac{\cfrac{\cfrac{(\text{mph})}{60} \times 5{,}280}{60} \times 12}{39.37} \times \text{length of animation in seconds}$$

4. Move vehicle M meters, then keyframe it as last keyframe.
5. Compute wheel circumference using the formula:
   $C = 2 \times \pi \times \text{radius}$
6. Compute pitch rotation of each wheel to match vehicle speed with the formula:
   $D = [(M/C) \times 360] \times (\text{length of animation in seconds})$
7. Set the pitch rotation of each wheel to D and keyframe as last frame.
8. Repeat for each wheel.

After I completed the movement of the Hummer, including the robotics of the guns and the missiles firing, Scott worked his magic on the desert environment. After that, it was just a matter of performing a Load Objects (Objects Panel) from the moving Hummer scene into the desert scene.

Joe Dox and Scott Wheeler operate a successful East Coast animation house called Galaxy Video & Animations. Together, this duo has created a very impressive demo tape. To contact them for suggestions or comments, or to obtain a copy of their demo, call GVA at (508) 535-8787; send e-mail to jdox@fastech.com (Joe), ord@dsn.com (Scott).

Mighty Morphin' Morphing Tricks

continued from page 13

- Do the same for "Box3" and "Box4," except move the box to the top left and top right areas of the Face view, respectively.
- Load all four boxes into Layout.
- Set **Object Dissolve** to 100% for Boxes 2, 3 and 4.
- Set the **Metamorph Target** for Box1 to Box2, Box2 to Box3, and Box3 to Box4.
- Make a wild, wacky metamorph envelope for Box1, Box2 and Box3 (Figure 3). Give each envelope a different total number of frames: something like 60, 52 and 46 frames, for example.
- Activate **Repeat** for each envelope.
- Make a wireframe preview that lasts for 300 frames.

Since the repeating morph envelopes overlap and are of different lengths, the motion of the box will not repeat for quite a number of frames, and you will get a sort of pseudo-random movement of the box as it flits about. Also note that the first envelope, for Box1, has more influence than the following envelopes. So too for Box2, and so on. Thus you might want to adjust the envelopes so that those earliest in the chain have a smaller range of values than the latter envelopes. For example, 40% to 60% for Box1, 20% to 80% for Box2, and 0% to 100% for Box3. This will tend to distribute the influence of the morph targets more evenly and the box will tend to hover around the center of mass of the four boxes.

Morphing Particles

Remember how I said there are exceptions to the rule about the target object having to be modeled directly from the source object? Here's an interesting one:

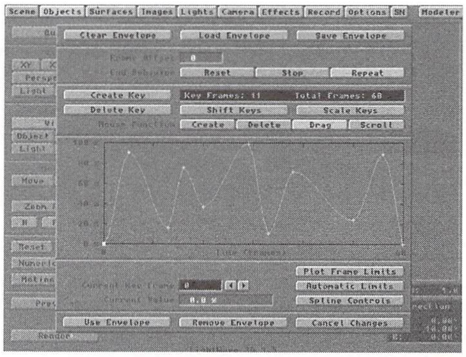
The effect is that of a cloud of several thousand stars drifting past the camera, *Star Trek*-like. Then the stars all change direction and race toward each other, congealing into the shape of a series of letters, or even a logo.

- First create some text in Modeler. Use the **Text** button (**Objects** menu or W).
- Load the Olnova Heavy font and type "HELLO." Click on **OK**.
- Center the text with the Center macro.
- Write down the size of the rectangular area occupied by the text plus a bit. For instance, if the maximum X value of the text is 1.64, write down 1.7 for the maximum X. If the maximum Y value is .39, write down .4 for the max Y. Approximations are fine.
- Go to an empty layer and put the text in a background layer.
- Choose the Point Spread (Point Distributions) macro and click on **Square** and **Constant**.

As a test to see if your size settings are OK, enter 100 for the Number of Points, and then enter the max X, max Y values that you wrote down into the first two Radius boxes below. Put a zero (0) in the third box.

- Make sure that Units is set to "m" for meters and click **OK.**

You should get 100 points arranged in a rough rectangular shape over the letters in the background. This macro actually creates single-point polygons, or particles, thus saving a step.



Frame Offset 0

Key Frames: 11 Total Frames: 60

Current Key Frame 0

Current Value 0.0 %

Figure 3: Example of a wild, wacky envelope.

- If the size of the area covered by the particles is correct, delete the 100 particles, re-enter the Point Spread macro and do the same thing with 10,000 points.
- Go get a soda—this will take a while. It took about five minutes on my machine.
- When the Point Spread macro is done, you'll have exactly 10,000 particles.
- Click on the **Copy** (c) button.
- Rotate the particles 180 degrees around the Y axis.
- Click on the **Paste** (v) button.
- You now have 20,000 particles.
- Choose **Drill** (**Tools** menu).
- Click on the **Z** Axis and the Core button. Then click **OK**.

A couple of minutes later, you should see a reduced number of particles in the shape of the word "HELLO."

- Click on the **Polygon** button and then use **Stats** (**Display** menu or w) and write down how many particles remain. I had 10,983.
- Save this object as "Letters." This will be the target object. Considering the size of it, you might want to delete all the polygons before saving. LightWave only looks at the points of a target object; polygons are irrelevant and just use up additional space on disk. [Editor's note: For morphs using **Morph Surfaces**, you will need to leave your polygons on the target object.]
- Clear this layer (z).
- Use the Point Spread macro again, except set the Number of Points to the amount you just wrote down, and set the Radius boxes to 10, 10 and 10. Again, make sure your Units are set to meters. Click **OK**.
- Go have another soda, or go to the bathroom because of the last one.
- Save the resulting object as "PointCloud." Do not delete the polygons, as this is the source object.
- Return to Layout and load both objects.
- Set the Letters object to 100% **Object Dissolve.**
- Set the **Metamorph Target** for PointCloud to Letters.
- Make a metamorph envelope for PointCloud with a keyframe at 0 equal to 0% and a keyframe at 60 equal to 100%. Make the Tension equal to 1.0 for both keyframes, while you're at it.
- Set the camera position to 0, 0, -6.
- Make a wireframe preview from 1 to 60.

This is just a small sampling of the many uses for object morphing. Of course, with Bones, you can do many of the tasks once done with morphing much easier and with more control (though Bones have their own limitations). I didn't even touch on Surface Morphing, which would require a fairly involved article all by itself. One day you might find yourself scratching your head over a particularly difficult project, and who knows—maybe morphing an object here or an object there will provide part of the solution.

James G. Jones is an independent animator whose business, Nibbles & Bits, is located in Colorado, where the air is clear and the oxygen insufficient. He can be reached via e-mail at jgjones@usa.net.

Popular 3D Algorithms

continued from page 11

other objects to the surface point in question. What is needed is a different method to handle this problem.

Radiosity

More of a pre-processing technique, radiosity deals with diffuse inter-reflections based on the physics of **energy** transfer. By breaking down the objects within a scene into a sufficient number of **elements** (polygons), you can calculate the amount of light that is being diffusely reflected by every polygon to every other polygon, thereby simulating the realism of diffuse inter-reflections. **Radiosity** simply refers to the overall amount of light leaving each element or polygon.

One of the factors necessary for achieving an accurate calculation of radiosity lies in effectively subdividing the objects into the proper number of elements. Too few may cause insufficient detail and too many may unnecessarily increase rendering time. Once the subdivision factor is determined, the algorithm proceeds in two major steps.

First, some idea of how much each element is visible from every other element is calculated mathematically. These **form-factors**, as they are called, are basically a percentage, or value between 0 and 1, determining how much of each element can be seen by another element. In other words, the form-factor expresses numerically the geometrical orientation of angles and distances between each and every element as a basis for accurately assessing the resultant energy (light) transfer.

For example, if two elements (read: polygons) are parallel and proximate to each other, the form-factor associated with this pair will be nearly 1. If, however, they are perpendicular to each other, the form-factor might be closer to 0, since very little of each element's surface can be seen by the other. Any other orientation of the two elements begets a form-factor somewhere between 0 and 1. Additionally, the form-factor is inversely proportional to the distance between elements. Once the form-factors for all elements is calculated, the second stage of the radiosity process can begin.

The algorithm then proceeds with a **balancing** loop to find the total radiosity (light energy) emanating from each element. This radiosity consists of both emitted (if a light source) and diffusely reflected light. By using the form-factors calculated earlier, this procedure eventually reaches equilibrium, whereby the energy absorbed and radiated by each element becomes balanced and further energy transfer between elements ceases. Perhaps this balancing process would best be understood by examining the mock pseudocode illustrated in Figure 3. In this very simplistic example, assume that there are only three elements and that one of these elements happens to be a light source. It might be helpful to imagine the walls and ceiling in the color image "Room Corner" as representing the variables in the pseudo-code, with the ceiling acting as a light source.

As the code shows, first the form-factors are calculated. Then the separate components of an element's total radiosity, the emittance and reflectance, are calculated. Only those elements that are actually light sources emit any energy, so that intensity is found for A, but the emittance of elements B and C are always zero. Secondly, the reflectance of each element, the light that is absorbed from other elements and redirected, depends upon the radiosity of the other elements modified by the form-factor, the physical orientation between the two elements. Since this is the initial phase of the loop, the radiosities of all the elements have yet to be calculated, so we approximate the reflectances with zero. Now the first estimates of the elements' radiosities are calculated — the summation of the emittance and reflectance variables. Finally, the code repeats this process of adjusting the reflectances based on the new radiosities as many times as necessary until an element's incident light equals its reflected light.

When the system has **converged**, the radiosities are then used in the final shading calculations to add some valuable extra color produced by the diffuse inter-reflections within the scene. Furthermore, besides the diffuse, specular and direct illumination now accounted for, shadow effects like penumbras and umbras are automatically handled by the form-factors. Not only are the direct shadows from light sources incurred, but shadows from other objects are inherently natural as a result of the form-factor calculations. At the cost of rendering time, the realism of an image can be greatly enhanced with the use of these subtle effects. It will be a great day when desktop processors become fast and inexpensive enough to feasibly handle these awesome rendering features.

C-Ya...Bye

I certainly hope this material has increased your appreciation of the nature of current and future 3D software. I'd hate to think of all you successful animators out there who have no idea of how your tools actually work underneath those fancy GUI veneers. It'd be kind of like a painter not knowing the subtleties of how one pigment might react with another when mixed together. Well, maybe not that extreme, but I think you get the rendering.

William is president of Ecliptic Arts, a developing 3D animation production house, and is also currently authoring a monthly graphics column for an Amiga-related publication called Amazing Computing. *He is considering changing his name to simply "Bilfro," as this might procure much-sought-after membership in the elite-status-one-nickname clique inhabited by such greats as "Sting," "Cher" and "Mojo." Send questions, comments or frozen pizzas to Ecliptic Arts c/o William Frawley, 315 W. Fifth Street, Muscatine, IA 52761.*

NAB MultiMedia World

CONFERENCE: APRIL, 9-13, 1995 • EXHIBITION: APRIL 10-13, 1995
LAS VEGAS HILTON • LAS VEGAS, NEVADA

Helping Business Profit from Multimedia

The Interactive Multimedia Association and the National Association of Broadcasters have assembled the best minds in multimedia to help you take advantage of current and future multimedia opportunities. NAB MultiMedia World offers strong, practical advice on integrating multimedia into your business and how to make it profitable!

THE EXHIBITION

Explore the industry's most comprehensive showcase of multimedia technology, products and services from more than 100 of the world's leading multimedia developers and manufacturers.

THE CONFERENCE

Gain the knowledge and the skills you need to compete and perform in multimedia, video production and broadcast communications.

ADDED BONUS!

Maximize your production or animation system's performance in the ALL NEW Light Wave 3D training classes and *Video Toaster User* Pavilion, #117.

For More Information

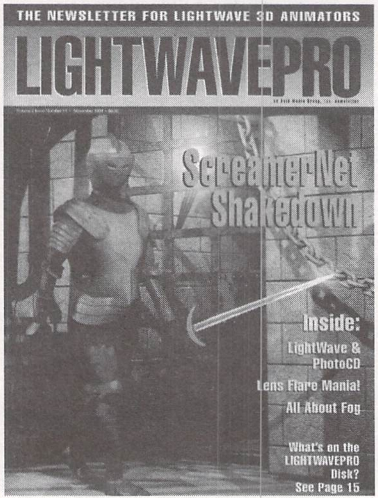
CALL
1-800-342-2460

FAX-ON-DEMAND
301-216-1847
CALL FROM THE HANDSET OF YOUR FAX MACHINE

INTERNET
<register@nab.org>

REGISTER TODAY!

LWP


THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS
LIGHTWAVEPRO
ScreamerNet Shakedown
Inside:
LightWave & PhotoCD
Lens Flare Mania!
All About Fog
What's on the LIGHTWAVEPRO Disk? See Page 15

TO ORDER WITH YOUR VISA OR MASTERCARD CALL TOLL FREE!

1(800) 322-2843

LIGHTWAVEPRO

THE NEWSLETTER FOR LIGHTWAVE 3D® ANIMATORS

Make a Smart Investment

Subscribe Today!

Basic Rate: $72
Canadian/Mexico: Add $US12
Overseas: Add $US36
Allow 6-8 weeks for delivery.

Make checks payable to LIGHTWAVEPRO.
Prepayment required on all overseas orders.

YES! Enter my one-year (12 issues) subscription to LIGHTWAVEPRO at the Special Introductory Rate of only $48—that's 50% off the cover price!

Name____

Address______________________________ Apt. #______

City__________________ State ______ Zip ________

☐ Payment Enclosed ☐ Bill Me

BFOLW

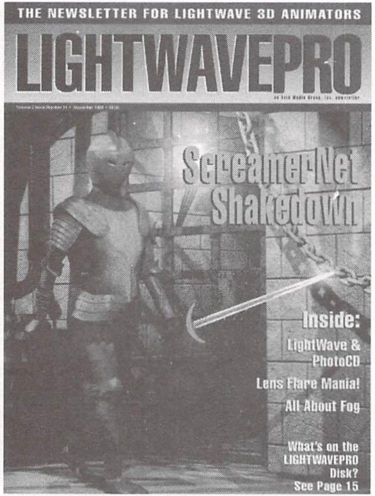
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

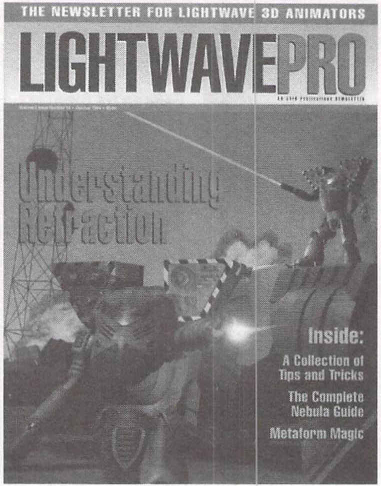
FIRST-CLASS MAIL PERMIT NO. 2263 SUNNYVALE,CA

POSTAGE WILL BE PAID BY THE ADDRESSEE

LIGHTWAVEPRO
273 North Mathilda Avenue
Sunnyvale, CA 94086-9313


THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS
LIGHTWAVEPRO
ScreamerNet Shakedown
Inside:
LightWave & PhotoCD
Lens Flare Mania!
All About Fog
What's on the LIGHTWAVEPRO Disk? See Page 15

**TO ORDER WITH YOUR VISA
OR MASTERCARD
CALL TOLL FREE!
1(800) 322-2843**


THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS
LIGHTWAVEPRO
Understanding Refraction
Inside:
A Collection of Tips and Tricks
The Complete Nebula Guide
Metaform Magic

TO ORDER WITH YOUR VISA OR MASTERCARD CALL TOLL FREE!

1(800) 322-2843

LIGHTWAVEPRO

THE NEWSLETTER FOR LIGHTWAVE 3D® ANIMATORS

Make a Smart Investment

Subscribe Today!

Basic Rate: $72
Canadian/Mexico: Add $US12
Overseas: Add $US36
Allow 6-8 weeks for delivery.

Make checks payable to LIGHTWAVEPRO.
Prepayment required on all overseas orders.

YES! Enter my one-year (12 issues) subscription to LIGHTWAVEPRO at the Special Introductory Rate of only $48—that's 50% off the cover price!

Name____

Address______________________________ Apt. #______

City___________________ State ______ Zip _________

☐ Payment Enclosed ☐ Bill Me

BFOLW

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

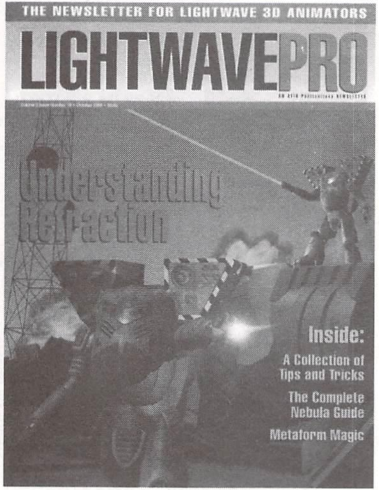
BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 2263 SUNNYVALE,CA

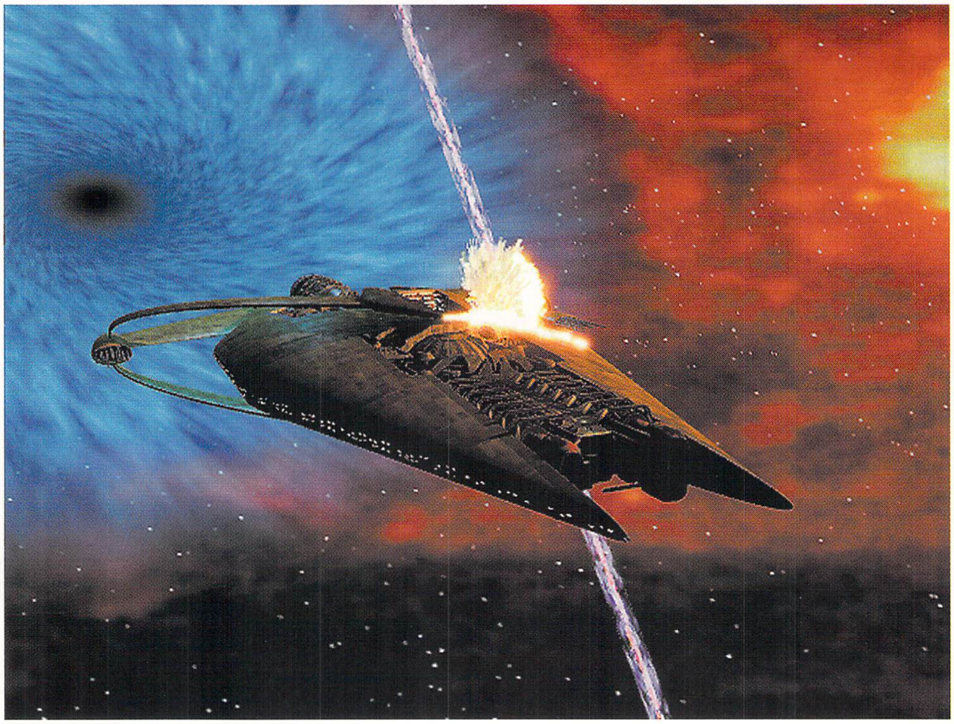
POSTAGE WILL BE PAID BY THE ADDRESSEE

LIGHTWAVEPRO

273 North Mathilda Avenue
Sunnyvale, CA 94086-9313


THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS
LIGHTWAVEPRO
Understanding Refraction
Inside:
A Collection of Tips and Tricks
The Complete Nebula Guide
Metaform Magic

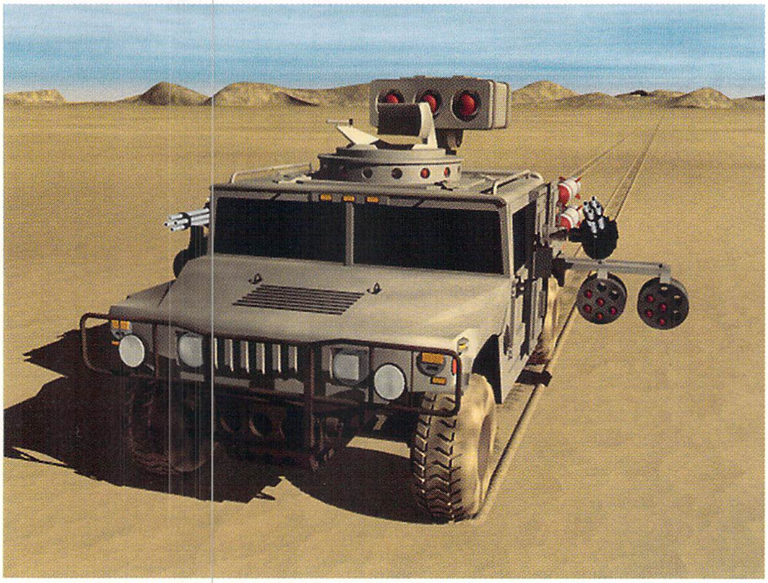
**TO ORDER WITH YOUR VISA
OR MASTERCARD
CALL TOLL FREE!
1(800) 322-2843**



Narn Cruiser

The beam slices through the ship. It contains a moving fractal noise transparency layer that quickly moves in the direction it's firing, giving the illusion of speed. See "Dead in Space," page 6.

Copyright 1994, 1995 PTEN Consortium, Inc.



Full Metal Hummer:

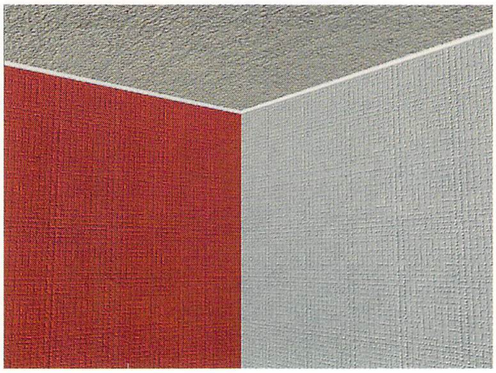
The Hummer object was modeled by Joe Dox of Galaxy Video & Animations. The Full Metal Hummer scene contains over 56,000 polygons, realistic motion, gun and missile launcher robotics, and, thanks to Scott Wheeler, a very nice desert environment. Joe and Scott can be reached at (508) 535-8787. See "Land Vehicle Movement," page 4.

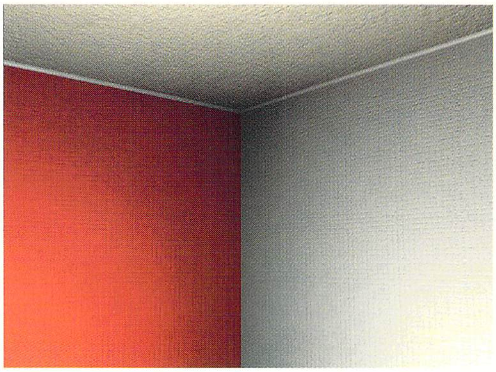
Copyright © 1994 Galaxy Video & Animations.

Room Corner

For more advanced applications, diffuse inter-reflections can be faked with the proper use of lighting. Instead of relying on the distant light source (A), point light sources with the appropriate falloff produce much more realistic shading (B). See "Popular 3D Algorithms," page 10.

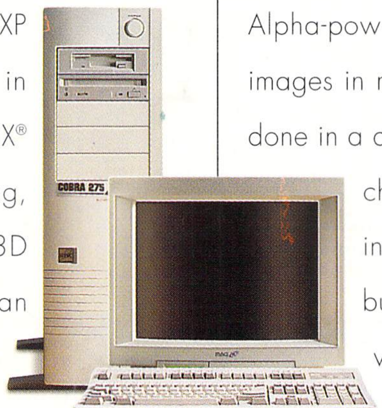
Copyright 1995 William Frawley.





RENDER GRAPHICS AT THE SPEED OF ALPHA ON A COBRA AXP 275 WORKSTATION.

ALPHA GENERATION™ There's no better way to burn graphics in LightWave™ or other applications. Introducing the Carrera Cobra AXP 275, the workstation leader in price and performance. Run UNIX® and Windows NT™ frame rendering, animation, multimedia and 3D graphics applications faster than you've ever seen on the power of a 275MHz Alpha™ processor—one of the 64-bit RISC rockets from Digital Semiconductor, a Digital Equipment Corporation business. With the blistering performance of an Alpha-powered Cobra, you'll generate digital images in minutes instead of hours. You'll get more done in a day. Maybe even get home on time for a change. And the Cobra AXP 275 comes in a variety of configurations, loaded with built-in PCI SCSI-2, PCI Ethernet, PCI video, PCI and ISA slots, CD-ROM, and more. Call or E-mail us for details. Then get ready for a workstation that really cooks.



CARRERA
Computers, Inc.

23181 Verdugo Dr., Building 105A, Laguna Hills, CA 92653 • 800-576-7472 • e-mail: CARRERA1@DELPHI.COM

© Carrera Computers, Inc. 1994. Digital, Alpha, and AlphaGeneration are trademarks of Digital Equipment Corporation. Other names are trademarks of their respective holders.